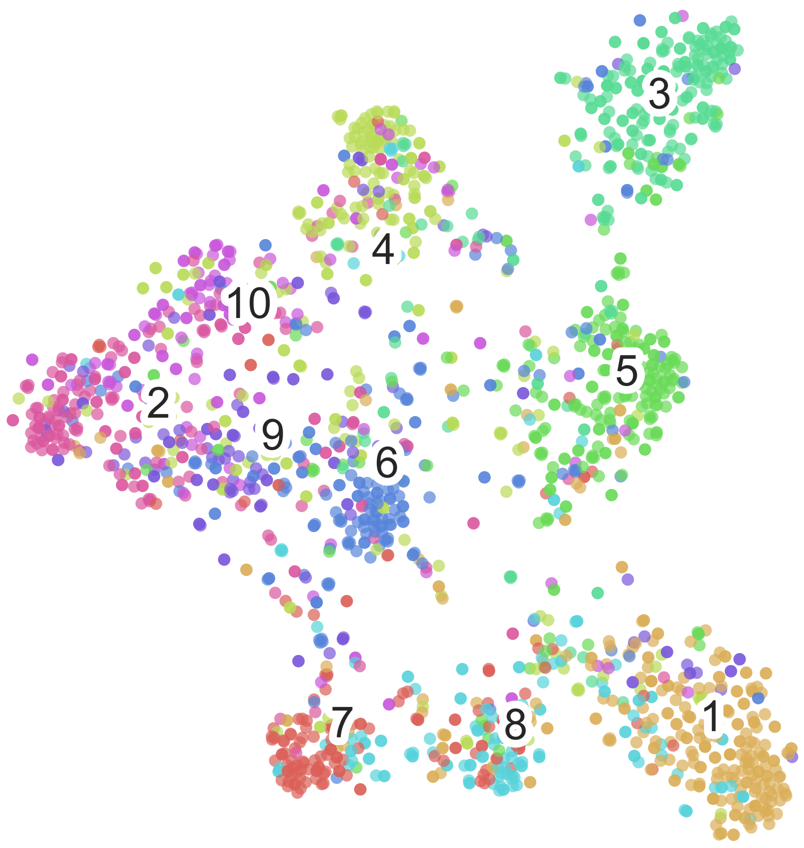
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Document Embeddings for Scientific Publications

Master's thesis in Software Engineering

FLORIAN SCHÄFER

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

MASTER'S THESIS 2018

Document Embeddings for Scientific Publications

FLORIAN SCHÄFER

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Supervisor: Hang Yin, CSE
Supervisor: Alessia Knauss, CSE
Advisor: Victor Botev, Iris AI AS
Examiner: Robert Feldt, CSE

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The embeddings for ten different topics of scientific documents have been projected into an two-dimensional vector space. Each topic is indicated both by its color as well as its number which is placed at the center of the topic's cluster. For more details on the figure, see section 4.4.

Abstract

While more and more research gets published today, it is consequentially getting harder for humans to keep up with new results at the pace at which they emerge. Even traditional computational methods cannot sufficiently handle such large amount of information which motivates us to utilize state-of-the-art research in order to find such a method that is first and foremost accurate enough to aid researchers in finding relevant literature while at the same time being computationally efficient enough to handle the increasingly larger amounts of data.

In this work, we specifically focus on vector space models since they enable us to utilize several efficient geometric computations. We first establish an evaluation framework including several metrics to be able to make a sounds assessment. Then, we explain and evaluate several neural network-based vector space models in the context of scientific publications using our framework. We thereby assembled two novel datasets for both the large-scale multi-domain corpus of Iris AI AS, as well as for the systematic mapping study on autonomous vehicles conducted by Chalmers University of Technology, which serves as an example for a single domain. Lastly, we analyze how well our retrieved vector space models can aid researchers in conducting systematic mappings studies compared to traditional methods.

We thereby found that the evaluated approaches strongly vary in their quality. Some performed barely above the random baseline, indicating either a lacking suitability of the method or being due to a lack of sufficient data or optimal hyperparameters. Especially sequential approaches and autoencoders, as well as the combination of the two, yielded surprisingly good results, which make these approaches worth considering for future studies. Apart from the quantitative results of our evaluation framework, we also provided a qualitative solution demonstrating for the autonomous vehicles mapping study how vector space models can provide benefits over traditional topic models.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In today's world, large amounts of research get published on a daily basis. This is especially true in the case of software engineering (SE). While the field was rather small in the beginning, it recently saw the advent of many new domains (e.g. autonomous systems, Internet of Things, smart systems) and conferences for specialized research areas (e.g. ASE for automated software engineering [1] or SSBSE for search-based software engineering [5]). A search for "software engineering" on the CORE online journal and research database [3] for example yields 1,121,115 articles (last accessed 23.02.17) out of which about 50,000 have been published in 2016 alone. As a consequence, single researchers and even teams can at best keep track of a rather small subdomain by continuously following that area. This makes it hard for people new to the field to get a general understanding of the topic based on the most eminent research results. Even harder is the challenge of connecting new findings across area borders.

Nonetheless, such challenges need to be dealt with when conducting systematic literature reviews [22, 52, 53] or mapping studies [71] in SE. If these limitations would not exist, people could surely perform such research or at least parts of it faster and more efficient. Furthermore, when getting insights into fields apart from the ones of the subjects main expertise, new interdisciplinary results might be retrieved that would not have been possible using traditional research approaches.

To be able to find relevant documents beyond the described human limitations, computers need to be able to aid researchers in the task of navigating through the large amount of potentially interesting research. Luckily, recent advances in the field of artificial intelligence and more specific machine learning have started to yield methods to compute the similarity between textual documents which potentially enables the goals described above. On the other hand and since this field is rather new, these methods are still not fully understood nor generally applicable.

The start-up Iris AI AS was founded in late 2015 to address the limitations described above, making use of these technological advancements. More specifically they work on creating a Science Assistant [6] that helps researchers to find relevant research amongst a large number of publications. Iris AI's Science Assistant can thus aid in finding and structuring the relevant literature as for example required by a systematic mapping study [71]. Even though a preliminary version of Iris AI's systems already exists, this thesis aims for improving the system by applying the latest state of the art methods in the field, leading to even more relevant results, while at the same time being able to retrieve those results fast among the millions of documents available. Furthermore, those methods might lead to insights on how documents are connected to each other, a feature not available in the current version of the Science Assistant.

1.2 Statement of the problem

While many approaches to compute textual similarity already exist, most of them focus on comparing documents in a pairwise manner. This way, when trying to find similar documents to a given document, the similarity between this document and each other document in the corpus has to be computed. Needless to say, this pairwise comparison does not scale to the millions of research papers mentioned in the introduction. A much more efficient approach lies in using nearest neighbor search [4](NNS) for which documents need to be projected into the vector space in such a way that any given document lies closer to similar documents and further away from unrelated documents (for more details see section 2.1.1). These vectors, obtained from the projection, are called document embeddings since they embed the textual information of the document in a semantic vector space. Document embeddings (also synonymously called document vectors or more general document representations in the following) are the foundation of vector-based document retrieval.

To obtain such document representations, Iris AI uses a heuristic function based on a number of methods including TF-IDF [76], NTM [23] and Word2Vec embeddings [63]), which are combined and weighted. While this approach is already quite effective, it still ignores many characteristics of the considered documents like word order or the document's structure which potentially could aid in better grasping a text's underlying semantics.

An alternative way of generating the required vector representations is using neural networks (NN) to create entire document embeddings at once [57]. Today many different approaches using various NN architectures exist (see section 2.2 for details), each with their own strengths and weaknesses. For many of them, it is unclear how well they perform on the task of creating document embeddings that aid in finding relevant research articles. Furthermore, the performance of NNs heavily depends on the structure of the input data as well as on the chosen hyperparameters which usually have to be determined empirically using a combination of expertise and extensive experimentation.

Another problem then lies in evaluating the quality of the previously mentioned approaches. While all publications describing such methods come with an evaluation part, it is still hard to compare approaches against each other. This is mainly because methods are rarely evaluated on the same task or the same metric using the same dataset. Needless to say, certain evaluation methods or datasets might favor the results of certain approaches over others, thus rendering them useless for a sound comparison.

1.3 Research Questions

RQ1 How can embedding methods be compared in a qualified and fair manner?
RQ1.1 What metric allows for a sound assessment of document embedding?
RQ1.2 What dataset(s) should be used that is not biased towards any method or task?
RQ2 How can scientific publications be transformed into document embeddings that work best for computing their semantic relation without being biased towards specific metrics or datasets?
RQ2.1 How well do different state of the art NNs for document embeddings perform on scientific publications?
RQ2.2 How do parameters like the length of the text to be transformed or the document vector dimensionality affect the quality of the resulting embeddings?
RQ2.3 In what chunks (characters, words, etc.) should text be fed into the NN?

1.4 Purpose of the study

The purpose of this study is to methodologically determine the best method to find meaningful vector space embeddings for scientific publications. The resulting embeddings will then allow using the usual location properties of a vector space. This mainly means that similar documents can efficiently be retrieved using their coordinates. Furthermore, analyzing the clustering of document embeddings within the vector space will most likely give insights into how documents, as well as groups of documents, are related to each other. This can for example aid researchers in conducting systematic mapping studies, where not only relevant literature needs to be found but also classified and structured [71]. One such mapping study is for example currently being conducted at the Department of Computer Science and Engineering at Chalmers University of Technology in Gothenburg, Sweden in the area of autonomous vehicles.

This will first of all benefit Iris AI AS's process of finding relevant literature since document embeddings are not utilized by them at the moment. If an effective vector space model can be found, it could replace their method to compute similarity by simply using the distance between document vectors. Furthermore, Iris AI AS right now represents documents as matrices rather than vectors. Moving from a matrix representation to a vector representation would thus enable considerable performance improvements and also enable methods like the earlier mentioned NNS.

Additionally, the created evaluation framework can be generally used for assessing the quality of document embeddings for other cases than the ones depicted in the case study of this work (see the design section for details) by providing both the methodology as well as the associated dataset.

Ultimately, the results of this work, when applied, will benefit SE researchers in conducting systematic literature reviews and mapping studies regardless of whether they mainly stick to one domain like the automotive mapping study or connect multiple domains like the broad corpus of Iris AI allows for.

2
Theory

2.1 Vector Space Models

As briefly mentioned in section 1.2, current research has been able to utilize vector space models for various fields within the area of natural language processing (NLP). Since they enabled achieving superior performance compared to previous methods, they quickly became the predominant group of approaches in the field. At the same time, they are easier to comprehend than other methods like graph-based approaches, heuristic, etc. Consequently, we will limit the scope of this work on vector space models rather than older methods, as the former seems more promising to us with regards to performance and scalability.

2.1.1 Properties of Vector Spaces

With all textual units represented as points within a vector space, calculating the distance between two units can be done by using trivial mathematics. The two most common distance measures for pairwise comparison are thereby euclidean distance and cosine distance.

The cosine distance (see equation 2.1) measures angle from center between two points.

$$cos(\vec{a}, \vec{b}) = \frac{\vec{a} * \vec{b}}{\mid \vec{a} \mid\mid \vec{b} \mid} \tag{2.1}$$

The euclidean distance (see equation 2.2) on the other hand measures how strong the semantic goes into a certain direction.

$$\|\vec{a}, \vec{b}\| = \|\vec{a} - \vec{b}\| \tag{2.2}$$

Based on the distance functions described above, vector space models can be used to both create clusters using algorithms like k-nearest neighbors (kNN) and retrieve related vectors using nearest neighbor search [4](NNS) utilizing algorithms like ball trees [69], locality-sensitive hashing [13, 17](LSH) or the Fast Library for Approximate Nearest Neighbors (FLANN) [66].

2.1.2 BoW Vectors

The arguably simplest way to create embeddings for words is by using one-hot vectors. These vectors contain as many rows as the vocabulary has words and, e.g. to represent the first word of the vocabulary, have the first row set to 1 and all other rows to 0. This kind of representation also generalizes easily to larger textual units like sentences or documents where each row is either set to 1 if a document appears in the unit or its number of occurrences. Because documents are only represented by the set of words they contain, regardless of their order and other features, this model is mainly referred to as the bag-of-words model (BoW). Since rows of words that do not occur in the textual unit are set to 0, BoW vectors tend to be really sparse. They also have to deal with the so-called curse of dimensionality as embeddings can easily grow to a few hundred thousand dimensions for corpora with large vocabularies. To deal with this issue, many approaches for dimensionality reduction like SVD [32] or feature hashing [75] have been applied to embeddings. Furthermore, they are not able to capture the semantic similarity of words or deal with synonyms. E.g. house and mouse have the same Euclidean distance as house and building even though the latter ones are semantically much closer. Regardless of those issues, one-hot vectors have still been able to achieve considerable results in practice and thus been the industry standard for many years.

2.1.3 Distributed Vectors

For some time, researchers have started looking at a words context rather than simply its occurrence. The main idea was already defined by Harris in 1954 to "know a word by the company it keeps" [39]. While some more or less effective methods have been proposed thereafter, only recently so-called neural word embeddings have become the state of the art where representations are learned by a NN [19, 30, 63].

The main benefits of distributed vectors are that they can indeed deal with synonyms or generally semantically related words and furthermore have a much lower dimensionality. On the downside, the fact that they place words, that appear in similar contexts, close to each other in the vector space, also treats opposites like hot and cold as semantically similar. Also, embedding size is a hyperparameter that can lead to drastic variances in performance on different downstream tasks like predicting the topic of a document. While the basic intuition is that higher dimensional embeddings can capture more semantic nuances, this does not mean that they perform better in a practical setting. This means that the embedding size usually has to be empirically chosen which can easily lead to embeddings that only perform well on certain specific tasks. Nonetheless, a higher dimensionality results in both increased training and inference times and furthermore requires more data if overfitting is to be avoided.

Until now, the two most popular methods for creating embeddings on a word level are Word2Vec [63, 64, 65] and GloVe [70]. While the later released GloVe claims to produce embeddings superior to Word2Vec, the experiments of Levy et. al. actually show the opposite [59]. Generally, both methods are widely applied nowadays, especially since both provide pre-trained embeddings on large vocabularies, thus saving the effort to run the computationally expensive training to start with.

As the next bigger lexical units, sentences also created interest to represent them as embeddings. While some approaches simply treat them as an arbitrary number of words [57, 47, 51], others utilize certain grammatical features like parse trees [79, 40]. In the following, a number of neural approaches to creating vector spaces for documents are introduced. Baroni's evaluation shows that neural presentations perform generally better on a wide range of tasks compared to BOW vectors [16] which motivates our choice to only include neural approaches. They are mainly selected because they seem particularly promising for our case of dealing with large amounts of scientific publications. We thereby make the assumptions that no parse trees like used by Socher et al. [79] or Hermann et al. [40] nor other kinds of target labels like used by Tang et al. [80] or Wang et al. [82] may be utilized even though the latter claim that such labels improve their models' performance. The main reason to not include target labels it that for the larger scale datasets used in this work, no sufficient amount of labeled data exists.

2.2 Neural Networks

2.2.1 Feed-forward NNs

While already many sources exist that explain neural networks in great detail, we will nonetheless give a brief explanation of how feed-forward neural networks work as they can be seen as the foundation of all approaches used in this work.

Loosely inspired by the human brain, artificial neural networks are made of cells which are grouped into sequential layers whereby mathematically every later is a matrix that is multiplied by its predecessor. While each network has an input layer as its first layer and an output layer as its last, it furthermore contains a number of so-called hidden layers which can range from one to arbitrary many, only being limited by computational power (see Figure 2.1). Each hidden layer can thereby be imagined as a number of neurons, which in mathematical terms can be referred to as one weight matrix. While the number of layers can have a major impact on the model's performance, it has to manually be chosen by the developer, making it a so-called hyperparameter. And while the size of the input and output layers are determined by the kind of data that is used, the size of each hidden layer is also a hyperparameter.

To compute an output using the network, a simple feed-forward pass is utilized. The input will thereby be multiplied with the weight matrix H of the first hidden layer with usually some bias term b being added as well. The results of this computation will then be fed into an activation function f, determining the ultimate output of the first hidden layer. The same process will then be repeated for all subsequent hidden layers until the output of the last hidden layer can be regarded the model's final output.

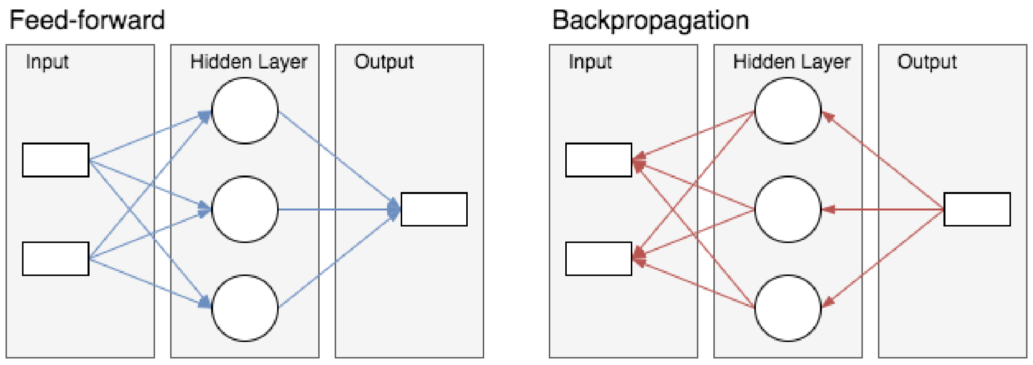


Figure 2.1: The feed-forward and backpropagation passes of a neural network.

While these weights are being somewhat randomly initialized for a new model, they will perform poorly unless they are tuned in the right way which is referred to as the model's training. The training of a neural network generally happens in two phases: In the first phase, the output for some training example is computed given its input data by using the previously described feed-forward process. In the second phase, the produced output is compared to the examples With the exact math being out of scope for this work, the error each weight has made with regards to the prediction is computed using a method called backpropagation [73]. The gradients computed in this process can then be used to make the necessary adjustment of the model's weights. This process is then repeated for a number of training examples until the model converges to an acceptable performance. When the training process is completed the network can predict the targets of additional inputs not contained in the training set by using the feed-forward procedure.

A high number of parameters might easily lead to overfitting which means that the neural network simply uses its weights to remember inputs and their respective target. While this behavior obviously leads to a good performance on the training set, it won't allow the network to generalize well, thus leading to a poor performance when being evaluated on the test data.

Another way to avoid overfitting is regularization. For this an additional term also referred to as regularization term, is added to the loss function of a model. This term imposes a penalty on the complexity of the model, thus making less complex models which tend to generalize better preferable. While many sophisticated regularization methods exist, the most fundamental and also most widely applied regularizers are L1 and L2 [68].

2.2.2 RNNs

Rather than feeding all words of a document into the network at once, recurrent neural networks (RNN) read document word by word. While reading a document this way, the network keeps track of an internal state that gets updated with each word that it reads. This provides the network with some notion of memory which enables the RNN to process every word it observes given a context based on earlier observed words.

Long-Short Term Memory (LSTM) models [42] are a special variant of recurrent neural networks (RNNs). The general idea behind RNNs for the creation of document embeddings is that they read through the document word by word while continuously accumulating a hidden state which memorizes the topical context based on the words read before (see figure 2.2).

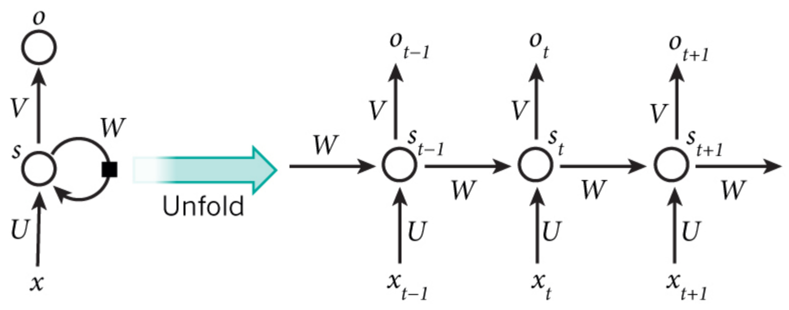


Figure 2.2: The basic RNN and its unfolded visualization as sequential steps.

While regular RNNs usually have to deal with the so-called "vanishing gradient problem" where the impact of previous time steps quickly fades for the current state of the NN, LSTMs are able to handle long-range dependencies much better. This makes them especially attractive for our case of producing embeddings for scientific publications of larger text length which might require the NN to remember the topical contexts across several sentences or even paragraphs. Also, Ghosh et al. [37] have applied this kind of architecture on the sentence topic prediction task (amongst others) with considerable success using both documents from Wikipedia as well as news articles from Google News, which makes their approach likely to also work for scientific publications.

2.2.3 CNNs

Another popular NN architecture is concurrent neural networks [58] (CNNs). CNNs became hugely popular in the area of computer vision (CV), especially after their successful application in the ImageNet LSVRC-2010 contest[1] by Krizhevsky et al. [54]. On the other hand, they are not that commonly utilized for NLP problems. One reason might be that for the visual tasks in CV they abstractly mimic the behavior of the human visual cortex which can be intuitively visualized [84]. For textual perception, on the other hand, the human reading behavior of looking at one word after another from one direction to the other (left to right in western culture) while remembering previously read words strongly equals how RNNs process input and thus makes it quite intuitive. CNNs on the other hand, which process information by sliding over the input data as explained below, appear way less intuitive.

[1] `www.image-net.org/challenges/LSVRC/2010/`

Nonetheless, a considerable amount of research for CNNs in the area of NLP exist.

Apart from common NN layers like the common softmax layer for classification tasks, CNNs are mainly based on two special types of layers, also shown in Figure 2.3: Convolution layers and pooling layers.

Convolution layers utilize a filter which is applied to the input data. The weights of the filter are thereby learned during the NNs training phase while the size of the filter and its stride (in what steps it is applied to the input) have to be defined as hyperparameters. Input data for CNNs can be either in form of the ordinal numbers of characters or embeddings for larger textual units like words or theoretically even sentences.

After convolutional layers, it is also common practice to use one or many pooling layers. This kind of layer subsamples its input, usually by taking the regional maximum or average. The main motivation of pooling is to reduce the layers output dimensionality in order to force the NN to only keep the most important information at this stage of the network.

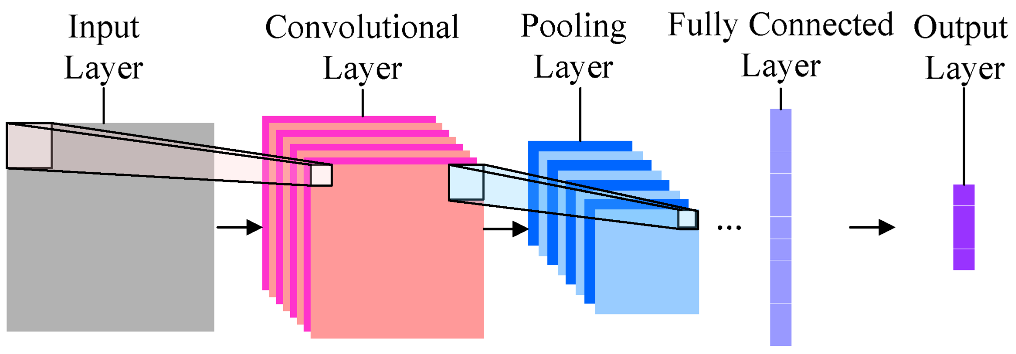


Figure 2.3: A general CNN architecture with one convolutional layer, one pooling layer and a fully connected layer at the end.

Some advantages of CNNs are that they are usually faster than recurrent neural networks (RNNs) and backpropagation is generally easier to implement. A considerable downside of CNNs, on the other hand, is the increased number of hyperparameters including the number and sizes of the convolution filters as well as pooling strategies.

Also, interesting variations have been proposed like Lai et al. [56] whose approach called RCNN combines both convolutional and recurrent NN architectures.

2.2.4 Autoencoders

While all neural network architectures presented above do usually rely on labeled training data, autoencoders take a somewhat different approach. Their way of working can be split into an encoding and a decoding phase (see Figure 2.4). For the encoding phase, they use a number of hidden layers to transform the input data into some intermediate representation. Thereafter, in the decoding phase, the model tries to recreate the input from the intermediate representation. The model's loss to be minimized is, therefore, the difference between the original and the recreated input.

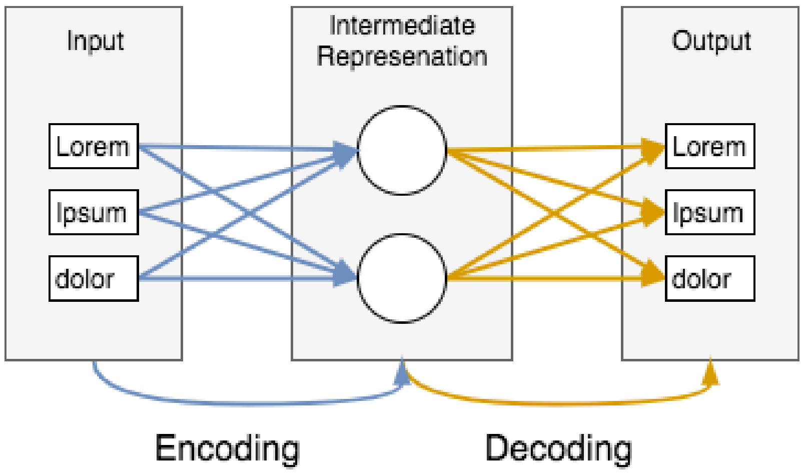


Figure 2.4: An autoencoder for word sequences using a one-layer feed-forward NN.

Since the intermediate representation is usually of lower dimensionality than the input, the autoencoder can be seen as a way to achieve compression or dimensionality reduction. Also, the hidden layers used for en- and decoding can be feed-forward NNs as well as CNNs, or RNNs.

2.3 Metrics

When assessing the quality of the document embeddings produced by the approaches described above, a sound metric is essential. Apart from enabling a fair assessment of different approaches, it also needs to be fast enough to evaluate a number of approaches within a reasonable amount of time. This can usually be achieved due to the fact that metrics are less computationally expensive than the actual embeddings methods as they do not need to learn a whole vector space model but only need to be able to pairwise compare embeddings.

The following section introduces some of the most popular as well as relevant metrics for our case of evaluating document embeddings.

2.3.1 Intrinsic vs Extrinsic Metrics

When assessing the neural language models described earlier, generally, two types of metrics exist:

Intrinsic metrics focus on comparing embeddings from a linguistic perspective. In the case of word embeddings, for example, it became quite popular to assess word similarity, where the semantic relation between words is compared [38]. Common datasets like *WordSim-353* [35] or *SimLex-999* [41] are manually curated and thus provide only a few hundred examples each. Le et al. [57], Dai et al. [31] and many others evaluate their methods based on "coherence", for which they check if a candidate vector lies closer to a manually picked related vector than to an unrelated one. The same procedure was also proposed by Schnabel et al. [77]. More generally the field of Semantic Textual Similarity (STS) [11, 12, 9, 8, 10] exists which aims for determining the semantic distance between two textual units. Unfortunately, most work in that field focuses on sentences which are usually represented by the average of their word vectors, while document-specific features are not considered.

Extrinsic metrics, on the other hand, assess the embeddings based on some downstream task like document class prediction. Since embeddings are usually created to be used in such tasks, it seems reasonable to also evaluate embeddings on downstream tasks, especially since Chiu and others claim that intrinsic metrics are poor predictors for an embeddings downstream performance [27, 60, 34]. Additionally, Nayak and Manning [67] propose a "Representative Suite of Practical Tasks" which promises a more thorough evaluation by applying several downstream tasks instead of only one specific task.

2.3.2 Supervised Metrics

Supervised methods are based on a definitive ground truth. Technically this means that for document embeddings, every document has to be labeled with its associated ground truth data also referred to as the target. The methods under assessment can then be evaluated based on how well it is able to infer the target based on the related document as input.

2.3.2.1 Topic Prediction

The possibly most popular downstream task in machine learning is the one of classification. In the case of scientific publications, this could, for example, mean predicting the category of a paper.

One example dataset for this kind of task is the Citation Network Dataset (CND) [25] which contains full scientific publications from the field of computer science including their abstracts. Each of the document was manually assigned one out of eight possible classes. Other examples are the Classic3 and Classic4 datasets created by the Cornell University [2] of which the later contains 7095 scientific documents manually labeled to four classes. Both of these datasets are suboptimal to evaluate approaches in the context of large datasets like the one Iris AI deals with, as the small number of classes will easily be exceeded in those scenarios.

Another unrealistic assumption is the existence of a fixed number of concrete topics. In contrast to clustering methods like k-nearest neighbors (kNN), the clusters in a vector space model are usually not bound by a specific number. This is mostly because documents are not clustered into concrete real-world topics but rather abstract dimensions learned by the model.

2.3.2.2 Information Retrieval

The information retrieval (IR) task takes one query document to then find the n most related documents to that one. This makes it particularly relevant for the use case of finding related publications.

Over the past decades, the IR community has developed several metrics to evaluate the effectiveness of retrieval algorithms. Those include Precision, Recall, F-measure, Mean Average Precision (MAP) and Normalized discounted cumulative gain (NDCG) to name some of the most popular ones [61].

The main problem with these rank-based metrics is that they require a manually curated ranked list of results for each query document to be tested. This creates an even larger effort then assembling a classification dataset as not only one ground truth label needs to be assigned but n ones instead. Also, the labels need to be ranked which introduces a certain subjectiveness when manually done by humans.

2.3.2.3 kNN Error

The k-nearest neighbors (kNN) error is similar to some of the IR metrics like precision and recall but more relaxed since the k most similar documents do not need to be ranked. Instead, it simply calculates the percentage of correctly retrieved documents out of the k candidates with the k-nearest neighbors from the dataset as the corresponding baseline. The main benefit compared to IR is that it somewhat removes the effort and subjectiveness introduced by the ranking.

2.3.2.4 Triplets

Another approach, also quite similar to the ones described before, is the triplets metric used by Le [57] and Dai [31] amongst others. Each of those triples consists of one "query" document, one "positive" document similar to the query and a third "negative" document different from the query. The goal is then for a given "query" document to compute a smaller distance to the positive document than to the negative one.

Like with the other supervised metrics, all triples have to be manually curated. While creating such triples is easier than n or k ranked or unranked neighbors, it might still not be feasible for large datasets.

What makes this metric especially appealing is that it assesses embeddings independent of any real downstream task, but rather focuses on inspecting single embeddings and their relation to another. On the other hand, the effectiveness of the metric again heavily depends on the subjective human judgment based on which the triples are assembled.

2.3.2.5 Curating Datasets

Unfortunately, supervised metrics also come with several downsides. To start with, they need to be curated manually. This inevitably introduces some degree of subjectivity originating from the person creating the data.

Also, this means creating larger evaluation sets can result in a considerable and often even unfeasible amount of work. For corpora in the size of 30M documents, this means to make a significant assessment at least 1.5 million documents need to be labeled with their respective ground truth. Needless to say, curating such datasets can quickly become unreasonably expensive, even when using services like Amazon Mechanical Turk [2].

Also, it needs to be guaranteed that the data is both comprehensive (i.e. covering as many words in the vocabulary as possible) as well as balanced (i.e. containing different types of documents in equal numbers).

As these downsides make using supervised metrics a large effort, not taking that effort means producing evaluation results of questionable quality, which nonetheless can be frequently seen in publications in the field.

Furthermore, it is questionable whether supervised metrics sufficiently assess the general quality of an embedding or rather its suitability for a specific task. One possible solution to this problem is, of course, to combine several metrics to an ensemble, resulting in a more general assessment but on the other hand increasing the effort even more.

2.3.3 Unsupervised Metrics

One possibility when sufficient ground truth data is not available is using unsupervised metrics instead of supervised ones. Rather than assessing documents based on their target labels, the method is given or learns some kind of distance function, based on which it regards documents more or less similar to each other.

2.3.3.1 Perplexity

The perplexity measure has its roots in information theory and is one of the oldest and most commonly used metrics when assessing language models. This is mostly because perplexity is intuitively easy to understand and can at the same time be computed efficiently.
Jurafsky defines perplexity as "the inverse probability of the test set, normalized by the number of words" [46] (see equation 2.3):

$$PP(w_1...w_N) = \sqrt[N]{\frac{1}{P(w_1...w_N)}} \tag{2.3}$$

Where:

[2] `https://www.mturk.com/`

- $P(w_1...w_N)$ is the probability of the words $w_1...w_N$ to appear in sequence
- N is the total number of words

Yet, in recent literature perplexity is broadly considered a bad measure. This is mainly due to its insufficient correlation to downstream tasks, which the assessed language model is developed for in the first place. It was even shown that the correlation between perplexity and human judgment is quite low as well [26]. Nonetheless, it remains widely used, possibly due to the lack of other more appropriate unsupervised metrics.

One attempt to specifically improve the correlation to downstream tasks is called Contrastive Entropy [14]. Unfortunately, the effectiveness of this approach is questionable and requires further evaluations which are beyond the scope of this work.

2.3.3.2 Embedding-based metrics

An intuitively more suitable approach for assessing document embeddings based on their distance is approximating that distance by comparing it to other embeddings. As the only purpose of such embeddings is to compute their pairwise distance, they do not need to form a vector space.

The easiest way to do so is by simply taking the average of all word embeddings of a document. In a second step, the Pearson correlation coefficient (PCC) (2.4) between two averaged vectors can then be computed.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2.4}$$

Where:
- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

The biggest problem of simply taking the average is that words that generally appear in various documents are considered equally important as words that are higher specific for certain documents only. One common way to deal with this issue is weighting terms using the term frequency-inverse document frequency (TF-IDF) statistic (2.5). This way terms are considered less significant the more documents they appear in. In case of document representations, this means that each word embedding for a document is weighted using the TF-IDF statistic of its term before all weighted embeddings are summed up in the end. Finally, the PCC can be calculated as for the averaged vectors.

$$w_{t,d} = tf_{t,d} * log(\frac{N}{df_t}) \tag{2.5}$$

Where:
- $w_{t,d}$ is the weight of term t occurring in document d
- $tf_{t,d}$ is the number of occurrences of term t in document d
- N is the total number of documents
- df_t is the number of documents containing term t

One general advantage of TF-IDF-based approaches is that they are really efficient to compute. When using well pre-trained embeddings, costly training can be avoided. Also, they are historically proven to work and are successfully applied in many NLP scenarios. Alternatively euclidean distance (equation 2.2) and cosine distance (equation 2.1) can also be used with TF-IDF vectors.

One downside of this approach is that it yields very high dimensional vectors of the size of the vocabulary, where most entries do not provide any useful information but only the absence of words in the respective document.

A similar yet slightly different approach Iris AI AS is using is their so-called Word importance-based similarity of documents metric (WISDM). Instead of accumulating the word vectors to one single vector to represent the document, they keep the vectors for all words within the document to then form a document matrix. To both save computational effort but even more, to filter out irrelevant terms, only words above a certain TF-IDF value are taken into account. For the resulting matrices, their pairwise RV[3] coefficient can then be calculated using equation 2.6. Since the RV coefficient can be seen as the multivariate generalization of the squared Pearson correlation coefficient, both methods are distinctly similar.

$$RV(X,Y) = \frac{Tr(X * Y^T)}{\sqrt{Tr(X * X^T) * Tr(Y * Y^T)}} \tag{2.6}$$

Where:

- Tr is the trace

On the downside, TF-IDF is not able to capture certain relations between words like word order. Nonetheless, Arora, Liang, and Ma report their TF-IDF-weighted GloVe embeddings to perform better than many more complex methods based on RNNs or LSTMs [15].

Yet, Wieting's experiments show that especially for tasks where the domain is not specified, these simpler unsupervised methods perform better than more complex ones like LSTMs [83].

Also, since most of these approaches are built on word vectors, it will in many cases be preferable to use pre-trained embeddings in order to save to computationally expensive training of those. One popular example for pre-trained word vectors is provided by Stanford, built using their GloVe algorithm, resulting in 300-dimensional embeddings which are trained on the 840 billion token Common Crawl corpus[4]. Another set of pre-trained vectors based on Word2Vec comes directly from Google itself and is trained on parts of their Google News dataset which contains about 100 billion words. The resulting model contains 300-dimensional vectors for 3 million words and phrases[5]. A third option, more specific to this work and also not publicly available, is the Word2Vec model from Iris AI. Their model was trained on 8 million

[3]`https://en.wikipedia.org/wiki/RV_coefficient`

[4]`http://nlp.stanford.edu/projects/glove/`

[5]`https://code.google.com/archive/p/word2vec`

random documents from their corpus containing a vocabulary of about 240.000 unique words.

2.3.3.3 EMD-based metrics

A somewhat more sophisticated approach is the so-called word mover's distance [55]. Unlike the previously explained methods, it skips the step of creating additional embeddings beyond the word level but instead considers document distance an optimization problem called earth mover's distance [72]. Extending this idea with supervision learning, Huang and others propose the supervised word mover's distance (S-WMD) improving test results even further [43].

With a complexity of $O(p^3 \log p)$, where p denotes the number of unique words in the documents, the WMD can be expected to scale rather bad for corpora with larger vocabularies like the ones Iris AI AS are dealing with. Similarly, solving the optimal transport problem for corpora with many documents will most likely be unfeasible as well.

3 Method

3.1 Applied Cases

The main goal of this study is to determine what methods to create document embeddings work best in the case of finding related documents from a collection of scientific publications. A number of potentially suitable methods will, therefore, be evaluated for two cases:

3.1.1 Systematic Mapping Study

The first and very specific case is an ongoing systematic mapping study on autonomous vehicles conducted by Chalmers University of Technology. The goal of this case is to find the most relevant literature within a fixed domain. It can thus be assumed that documents in this domain are generally quite similar for example regarding their vocabulary.

3.1.2 Iris AI

The second and much more general case is Iris AI's Science Assistant to find related literature in a broad variety of fields. In this case, documents will show a much greater variety in topics and vocabulary. Also, computational efficiency will become an important factor here since the methods under assessment have to scale to a corpus of several million documents. This case is more representative for literature studies that involve more than one domain like in the area of search-based software engineering where regularly research from non-SE fields like for example genetic programming is incorporated [18].

3.2 Evaluation Framework

To be able to make a sound and fair assessment of the quality of vector space models we need a unified framework. Only such a framework will allow a reasonable comparison of different models, much unlike the current status quo in research where most publications base their results on a broad variety of different datasets and metrics, making comparisons impossible without further self-conducted experiments. This section will, therefore, introduce our novel framework for the evaluation of vector space models in the area of scientific publications as well as an experiment using this framework to evaluate the performance of several state of the art models.

3.2.1 Evaluation Methodology

First, the effect of those parameter selections on the embedding quality will be assessed within each case individually. Thereafter, a cross-case analysis will be performed to see how findings from the individual analyses compare and generalize. Ultimately, it needs to be assessed how these insights can be used to obtain the highest quality document embeddings for both smaller single-domain cases as well as cases with large multi-domain corpora.

Instead of running ad-hoc experiments that are hard to compare or reproduce, we base our framework on a well-defined evaluation methodology. Hence, our framework is based on the following five steps, also shown in figure 3.1:

1. Select Approaches
2. Select Metrics
3. Select Data
4. Optimize Hyperparameters
5. Run Experiments

Unsurprisingly the first step lies in determining the approaches (i.e. vector space models) under evaluation. In the second step, one has to decide what metrics to use. To be able to compare experiments, we strongly recommend using our proposed metrics described in section 3.2.2. Nonetheless, further metrics can be added if the use case requires so. For step three a dataset to train and test the vector space model(s) on has to be appointed. Again we recommend using one of our datasets specifically curated for this framework. If the processed documents should come from another domain than scientific publications, it is also possible to operate on those datasets by applying our preprocessing pipeline, explained in detail in section 3.2.2.4. This way new datasets in the same format can be curated to then be used without making technical changes to the framework. One example use case for this could be the creation of a dataset specialized on medical publications taken from the PubMed repository [1]. In the fourth step, the hyperparameters of the model will be optimized. While this step is optional and might be skipped for the rare case of models without hyperparameters or the assumption that the hyperparameter space has already been sufficiently explored, it is usually necessary since most models strongly vary in performance based on their hyperparameter choices. Section 3.2.3 describes our preselection of common hyperparameters to be optimized. Needless to say, additional hyperparameters can be added based on the approaches under evaluation. At last, in the fifth step, the actual experiment(s) will be conducted by measuring the defined metrics on the approaches trained on the previously determined dataset.

3.2.2 Metrics

Rather than relying on a single metric, a small ensemble of conceptually different metrics is used to provide more sound and unbiased evaluation results. The challenge hereby lies mainly in finding an ensemble that can sufficiently assess document

[1] `https://www.ncbi.nlm.nih.gov/pubmed/`

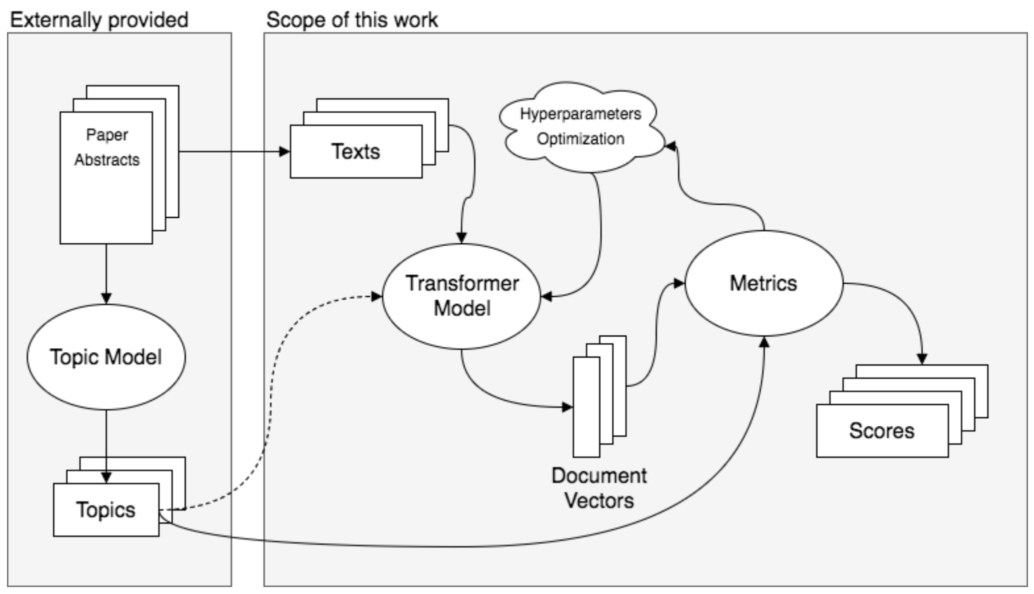


Figure 3.1: This figure shows the different steps in our framework as well as the respective information flow.

embeddings while at the same time being feasible with regards to computational efficiency. Also, while chapter 2 already gave some background on the applied metrics, this section goes more into detail how exactly those metrics have been implemented in our framework.

3.2.2.1 WISDM-based Distances

As a first metric, the triplets as used by Le [57] and Dai [31] will be applied. Therefore, a number of random document embeddings will be sampled from the vector space under evaluation. For each vector representing a document, another random point below a defined maximum distance and one above some minimum distance are sampled as well. Since this would mean that for every sampled document we would have to compute the distances to (approximately) all other documents within the dataset, we use precomputed distances based on the earlier mentioned WISDM similarity metric. Because this metric takes no other data than the learned document vectors under comparison into account, it can be seen as task-agnostic. It, therefore, aids especially in assessing embeddings in an isolated manner only based on some similarity function rather than measuring its performance on some downstream task.

Another metric where the WISDM distances can be used is the information retrieval task. For this, a number of documents are sampled from the test dataset and their n nearest neighbors are retrieved from the vector space model under assessment using NNS. In analogy to the triplets metric, we use precomputed WISDM distances to make the evaluation computationally feasible. Unlike the task-agnostic triplets, this metric is more related to the later application of the document embeddings where related scientific publications have to be retrieved ranked and in limited numbers.

3.2.2.2 NTM-based Topics

The goal of the topic prediction metric is to correctly infer the topic of a document given its full text or abstract. The main motivation for this metric is to present an alternative to the previously described word vector-based metric, avoiding its inherited bias from the pre-trained embeddings. While a generally superior vector space model is expected to perform best in both this metric as well as the previous one, divergences can give hints certain metrics are biased towards certain approaches.

Instead of manually labeling the documents, the target topics are taken from the Neural Topic Model [24] (NTM) of Iris AI which itself is trained in an unsupervised fashion and only refines its results using manually labeled documents. The benefit from not using the manually labeled data to start with is that the NTM can be utilized to label an arbitrary number of documents and thus allows to scale the test set size according to the size of the corpus that the approach under assessment should be later used on. The report results will then simply be the percentage of correctly classified documents using logistic regression for classification. While non-linear models might perform better on the classification task itself, linear ones are faster and still fair when used for all approaches.

To obtain the topics necessary for the supervised class prediction metric, the NTM from Iris AI will be used. In case of the Automotive dataset, it will assign one out of 200 topics to each document. For the Iris AI dataset, a document can be assigned to one of 100 topics. The number of topics was determined experimentally based on the best perplexity of the used NTM.

3.2.2.3 Datasets

Since the quality of any statistical model heavily relies on the data it is based on, we took special care of curating a number of datasets to be used in our evaluations. The fact that we created several datasets instead of one has several reasons: First, we obviously needed at least two datasets based on our two applied cases described in the beginning of this chapter. Second, while many NN models are too computationally expensive to be run on larger datasets, they might only be feasible when evaluated on a smaller dataset. Third and last, with only small datasets available in many practical applications, it makes a lot of sense to evaluate vector space models on a comparably small dataset for the fairest possible assessment. Similarly, for cases where the dataset can be expected to be of a larger size, evaluating models on a respectively larger dataset only makes sense.

The documents to evaluate the Iris AI case, are retrieved from the aforementioned CORE repository. Similar to before a NTM has been utilized to label X documents of the corpus with topics. This time, the best perplexity was achieved by using 100 topics.

To be able to evaluate the vector space's behavior on various kinds of datasets, we created three different datasets from the CORE repository:

	Core10k10	Core250k10	Core1M+100	AV5k10
# Documents	10000	250000	1022008	5018
# Sentences	77637	1957206	7935477	34463
# Words	969736	24326659	100033132	409080
# Labels	10	10	100	10

Table 3.1: The four datasets created for the evaluation framework.

Core10k10

First, the smallest *Core10k10* dataset consists of 10000 documents, equally distributed to the ten most frequent topics. We reduce the number of topics from 100 to 10 including the following topics IDs: *44, 99, 94, 19, 85, 93, 97, 49, 30, 20.* While *6* is the x most frequent topic, we manually removed it since this topic produced by the NTM appears to mainly capture documents of short nature. This results in the issue that many of this topic's documents cannot be used by models like Doc2Sent2Vec which require a certain document length or generally decrease the amount of information when being padded in order to guarantee a minimum number of words for a document or sentence.

Core250k10

Second, the medium-sized *Core250k10* dataset which draws from the ten most frequent topics, but this time 25000 documents each. Just like the *Core10k10* dataset all topics are balanced.

Core1M+100

Third and last, the *Core1M+100* which - like the name suggests - contains over a million documents from all 100 topics. More precisely, it is made of all 1022008 topic-assigned documents available in the corpus. Unlike the earlier two, the last dataset is not balanced with regards to the documents' topic distribution because of the small number of documents assigned to the least common topics.

AV5k10

Additionally, we curated another type of dataset for the autonomous vehicle case to which we refer as *AV5k10.* The study examines a total of 11414 research papers - or more precise - the papers' abstracts. While no human-labeled topics for those papers exist, a neural topic model (NTM) has been used to assign latent topics to the documents. The best perplexity and efficiency, when applied to mapping studies, were thereby reached with a number of 200 topics.

For the evaluation part of this work, we created one dataset composed of 5000 documents from the 10 most frequent topics. It is thereby worth mentioning that even for only 10 topics, those are not balanced mainly due to the fact that the documents are so widely spread over 200 topics, which leaves only a few hundred to thousand documents per topic.

3.2.2.4 Preprocessing

Our preprocessing approach can be divided into two passes:
In the first pass, a preliminary vocabulary is built by extracting all unique tokens from all documents within the corpus. Before going through all documents, the corpus gets shuffled to avoid any bias from the order in which documents were indexed in the repository. Then, for each document the following steps will be applied:

1. Clean title and text body.
2. Discard documents with empty title or body.
3. Discard duplicates (= documents with the same cleaned title).

The cleaning step is itself made of several finer-grained steps which are explained in the following: First, documents will be tokenized into sentences. This step is mainly necessary for the Doc2Sent2Vec approach that processes documents on a sentence level. Second, all previously retrieved sentences will be further tokenized into single words. These words will then be lemmatized in order to reduce semantically equal words like synonyms to the same base token. In step four, non-alphanumeric characters will be removed from the tokens. The purpose of this measure is mainly to remove for example markup which contains no semantic information as well as all kinds of punctuation. This is desirable in our case since certain types of punctuation like exclamation signs or questions marks which can provide valuable information for tasks like sentiment analysis, are generally rare in scientific publications and furthermore do not aid in capturing the topic or content of a document. At last, we remove empty or single character tokens, as we make the assumption that single characters or letters contain no information regarding their respective document.
Furthermore, for each token in our corpus we

1. Collect the number of total occurrences,
2. Collect the number of documents containing a specific word,

which can be understood as building an inverted index like it is common practice for most textual information retrieval tasks.

Since the complexity of most language models heavily depends on the vocabulary size, we further reduce the vocabulary while at the same time removing tokens that do not provide information regarding a document's content: First, we remove all tokens that appear in 10 documents or less. We argue that these tokens are too specific with regards to certain documents while not being helpful to compare such documents to others. Second, we remove all tokens that appear in at least 10% of all documents. The motivation behind this reduction step is that such tokens are too generic, again not contributing to capturing a document's content. We chose this method over the usual stopword removal technique using general stopword lists like the Stopword Corpus by Porter et al.[2] because scientific publications contain a lot of words like study, research or result that are generally common within the field of scientific publications and while being helpful in a common knowledge setting where they would indicate a document being a scientific publication, we already make this assumption based on our corpus, thus gaining no information from such

[2] `http://snowball.tartarus.org/algorithms/english/stop.txt`

words. At last, all tokens will be checked if they exist in the reduced vocabulary and if so replaced by their respective position in the vocabulary (i.e. a numerical ID). Out-of-vocabulary words will be substituted with a shared *UNK* token. So when the whole vocabulary was built, we ended up with a total of 131873 unique tokens.

3.2.3 Hyperparameters

Apart from the hyper-parameters specific to each approach, a couple of general parameters have to be considered for the experiment as they are expected to have a considerable influence on the quality of the produced embeddings.

3.2.3.1 Embedding Dimensionality

While some sources claim that a higher dimensionality will yield more expressive embeddings, others show that after a certain number of dimension performance will decrease. Also, using higher dimensional embeddings results in both longer training and inference times as well as larger space for the model to be saved on hard disk. When using the model in memory it can even become unfeasible once the model size exceeds the memory capacities. Therefore we should always aim for smaller embedding sizes as long as the performance does not suffer.

3.2.3.2 Number of Epochs

An epoch is defined as the amount of training in which the model sees every example of the training data. After each epoch, the examples are shuffled to avoid fitting the model to the order in which examples appear. In our experiments, we generally use a high number of epochs combined with a technique called early stopping which stops the training when the model does not improve over a couple of epochs in order to reduce overfitting. Since stochastic optimization methods tend to sometimes stagnate or even become wo but then return to improving after, we utilize a patience count that will continue the training for n epochs without improvement until the maximum patience is reached.

3.2.3.3 Batch Size

The batch size determines how many training examples are looked at before the gradients are applied to the model's weights. A small batch size, therefore, results in many inaccurate updates, while a bigger batch size results in fewer updates but since more examples are taken into account, those updates can be expected to be more accurate.

3.2.3.4 Learning Rate

The learning rate determines how much of the calculated error gradients are applied to the model weights. While a too high learning rate might miss minima, a too small learning rate might get stuck in local minima or not converge in time at all.

For the models in this work, we chose 0.0001 as the lower and 0.1 as the learning rate's upper bound.

3.2.3.5 Optimizer

We have selected three of the most widely used optimizers:

First, Stochastic Gradient Descent (SGD) can be considered to most basic optimization method for neural networks. The stochastic term is based on the fact that the gradient is not computed for the whole number of training examples at once but rather in small batches where each batch already takes into account gradient updates made by prior batches.

Second, Adagrad [33] uses an adaptive learning rate that unlike the static one of SGD gets reduced for large gradients and increased for small gradients, thus avoiding the problems resulting from an inappropriate learning rate.

Third, ADAM [50] is also adaptive like Adagrad but additionally uses momentum where every batch does not produce a completely new gradient but rather adjusts the previous gradient into a certain direction.

3.2.3.6 Drop Out

One common trick to improve a model's accuracy is the usage of so-called drop out layers [36] which simply ignore a certain percentage of the previous layer's weights. The intuition behind this method is to improve generalization by forcing the network to not rely on single neurons for prediction.

In our experiments, we use drop out for all models except Doc2Vec and Doc2Sent2Vec. Our lower bound is thereby 0.0 which means no dropout and 1.0 which results in dropping all weights.

3.2.3.7 Hyperparameter Optimization

While selecting hyperparameters by hand is a long and tedious process, it seems only logical to automate this process. The most obvious solution is a method called grid search where certain combinations of hyperparameters will be evaluated. Unfortunately, when aiming to be exhaustive and working with more than just a few hyperparameters, this process will create a huge number of permutations which are almost always unfeasible to completely train the model on. More practical is an alternative called random search. Much like the name suggests, it simply tries out a number of random hyperparameter combinations and is widely accepted to produce much better results than grid search while only needing fractions of its time to compute [20]. One downside of the random search though is that it does not remember which prior combinations perform better or worse and thus might spend unnecessarily much time exploring parts of the hyperparameter space which can be expected to be futile after a few combinations.

Instead, we are using the particle swarm optimization (PSO) algorithm [48, 29] provided by the aforementioned Optunity framework. PSO optimizes a given function by trying out a number of candidate solutions also called particles. All particles are referred to as a swarm which iteratively moves towards the global minimum by following the best particles. While Bayesian Optimization [78] follows a similar approach, this method works best when good priors are available which is not the case for our scenario. Also, Bayesian Optimization implementations are much harder to execute in parallel, while Optunity proves a highly concurrent PSO implementation.

The function we chose to maximize is a weighted combination of the metrics described earlier (see Equation 3.1). Needless to say, while the topic prediction and triplets metrics are weighted with a third each, the accuracy and precision of the ranks metric are weighted a sixth each, making for a total third for the ranks metric, thus weighting all metrics equally.

$$score_{total}(m,h,d) = \frac{1}{3}score_{tp} + \frac{1}{3}score_{tr} + \frac{1}{6}score_{ra} + \frac{1}{6}score_{rp} \tag{3.1}$$

Where:

- m is the model under evaluation.
- h are the model's hyperparameters.
- d is the dataset used for evaluation.
- $score_{tp}$ is the accuracy of the topic prediction metric.
- $score_{tr}$ is the accuracy of the triplets metric.
- $score_{ra}$ is the accuracy of the ranks metric.
- $score_{rp}$ is the precision of the ranks metric.

As this score is rather abstract and thus not intuitively easy to interpret regarding the quality of the assessed model, we will establish a simple baseline based on random guessing and compared to which a model should improve. In case of the triplet metric, there are only two options to choose from which gives us a 50/50 chance of guessing the correct result. For the topic prediction metric, we have 10 equally balanced topics and therefore a chance by guessing of 10%. The chance of guessing the correct rankings, however, is negligibly small and can thus be assumed zero in our case. This gives us a random baseline of 0.18 as shown in equation 3.2.

$$score_{baseline} = 0.3 * 0.5 + 0.3 * 0.1 = 0.18 \tag{3.2}$$

In order to find an optimal combination of hyperparameters using PSO, we will run 5 generations with 7 particles each. While more particles and generations might be able to find better combinations of hyperparameters, this is unfortunately not feasible in our case considering the time restrictions of this work.

To keep the comparison fair, we keep the hyperparameter boundaries the same for all models: For the document embeddings size, we chose to sample from a range of 20 to 200, mainly because most publications report optimal embeddings sizes in that area. As it is rather hard to get an intuition for an appropriate batch size, we chose a rather wide range from 20 to 500. As optimizers, we limit our choices to the previously introduced three most common ones, namely SGD, ADAM and Adagrad. For the learning rate, we chose 0.0001 as the lower and 0.1 as the upper bound. The minimum delta for no improvement with regards to early stopping is 0.001 together with a patience count of 3. For the models that use dropout, we sample from the full range of values between 0.0 (keep all the weights e.g. no drop out) and 1.0 (drop all weights).

3.3 Experiments

In this section, we describe our experiment on the assessment of vector space models. The purpose of this experiment is mainly to understand how some popular approaches perform given certain datasets and hyperparameters. This will then help to decide what approach to use in practice and what hyperparameters to chose for them. Furthermore, it serves as a proof of concept for the feasibility and practicability of the framework and its underlying research methodology. Ultimately, it can even be seen as a blueprint for future experiments in this area.

3.3.1 Approaches

This section will introduce the used artificial neural network approaches utilized to create document embeddings. We provide both an explanation of how the respective algorithms work as well as a rationale why these approaches were chosen.
Rather than including the code for our implementations here, we aim to explain our models as detailed as possible. Since the landscape of deep learning frameworks is in rapid and constant change, source code written for a specific version of the framework is usually outdated and thus not working anymore within only a few months. Instead, we believe that our detailed descriptions can be used as more timeless blueprints that enable others to reproduce our results in any language or framework.

3.3.1.1 Doc2Vec (Paragraph Vector)

The two main downsides of the BOW model described earlier in section 2.1.2 is that it disregards both word order and semantics and thus loses a lot of textual information. While considering pairs, triplets or generally n-grams of words incorporates word order at least to some extent, the number of n-grams grows extremely fast for larger vocabularies and n values and furthermore creates large sparsity as usually, only a few n-grams occur within a given document.

To deal with this issue, Le and Mikolov, the main contributor behind the Word2Vec papers, extend the idea of word embeddings to larger textual units like sentences or even full documents [57]. To achieve this, they make use of an NN similar to the one previously used by them to trained word embeddings. For their more general approach called Paragraph Vector (PV), they move from single words to sequences of words which they refer to as paragraphs. A paragraph can thereby be a sentence as well as an actual paragraph or even a whole document like a newspaper article or a scientific publication.

The original Word2Vec method starts with transforming the variable-length words of the training corpus into numerical vectors of fixed length. This is done using a look-up matrix W with N columns where N is the size of the vocabulary and each column contains a vector of size E, where is the size of the embeddings which has to be chosen as a hyperparameter.

Furthermore, two variants of the algorithm exist: While the distributed bag-of-words (DBOW) variant uses a word sequence of fixed length as input to predict the word following that sequence, Skip-gram only uses a single word as input to predict the word sequence following that word using in both cases the log-probability defined by equation 3.3. For both variants, the weights of the NN are trained using stochastic gradient descent (SGD), with backpropagation [73] being used to compute the gradients. The resulting type of model is also commonly referred to as a neural language model [19].

$$y = b + Uh(w_{t-k} \dots, w_{t+k}; W) \tag{3.3}$$

Where:
- U, b are the softmax parameters.
- h is constructed by a concatenation or average of word vectors extracted from W.
- $w_{t-k} \dots, w_{t+k}$ are all words of the trainings set T within the context of size k.

For the PV approach, not only a word matrix W but also a paragraph matrix D is utilized. Similar to W, D holds an (usually randomly initialized) embedding of size E (which is again to be determined as a hyperparameter) for every paragraph in the training corpus. While h in Equation 3.3 was just computed from W in Word2Vec, for PV h is computed from both W and D. The paragraph token from D can be understood to memorize the context or topic of the current paragraph because of which this variant of PV is also called Distributed Memory Model of Paragraph Vectors (PV-DM).
To train the weights of the NN, each paragraph is split into contexts of fixed length, which then together with the paragraph token are used as input to the NN. The paragraph vector is thereby shared across all contexts. D, on the other hand, is not shared across paragraphs but W is which means that for each word the same vector is used across all paragraphs.

The context and paragraph tokens are then either averaged or concatenated and sent to the softmax layer. Again, similarly to the original Word2Vec, the weights of the NN are then trained by using SGD with backpropagation. When using the NN to perform inference, the word and softmax weights are fixed. The gradient for the unknown document vector is then computed to retrieve its embedding.

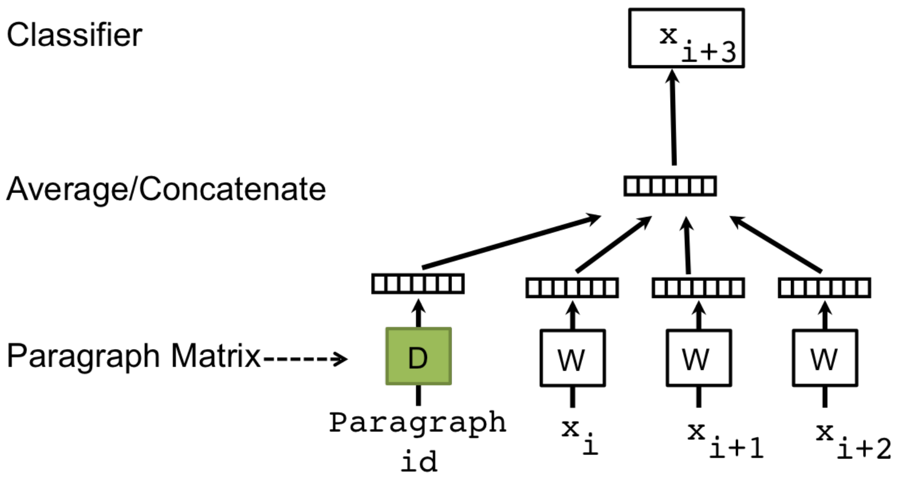


Figure 3.2: The distributed memory variant of PV. Both context and the paragraph token are used as input to the NN. [57]

Just like the Word2Vec method, a second variant exists that is trained on predicting a sequence of words. Analogically this variant is called PV-DBOW and uses only the paragraph vector to predict word vectors from the respective paragraph. Since only the softmax weights and no word weights need to be stored, this variant needs considerably less space.

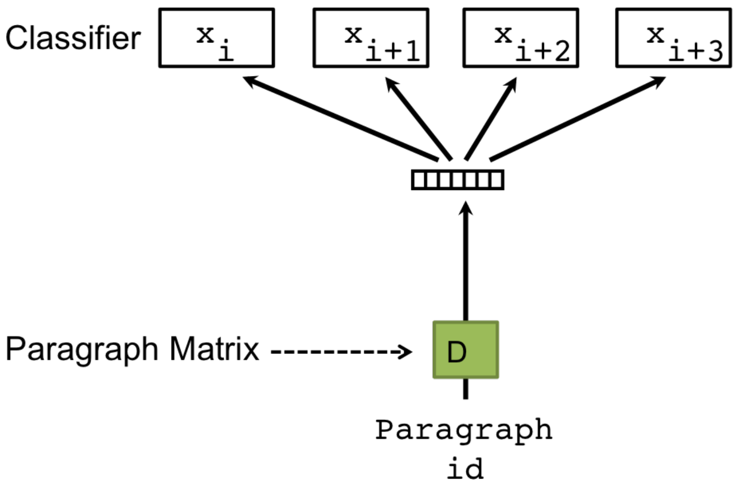


Figure 3.3: The distributed bag-of-words variant of PV. Only the paragraph token is used as input. [57]

PV is widely considered a strong baseline which is why we use it as our baseline approach as well. It is applied to many downstream tasks which makes it a fair baseline compared to more specialized approaches. Also, in their paper, Dai et al. evaluate PV on 4,490,000 articles from the English Wikipedia using a vocabulary of 915,715 words [31] which suggests that the approach scales to larger datasets like the one of Iris AI. Like many other publications that use PV, we also use the PV-DBOW variant due to its higher efficiency while being comparable in performance with PV-DM.

3.3.1.2 Doc2Sent2Vec

Similar to how the PV method used by Dai et al. jointly learns word and paragraph vectors [31], J et al. propose a variant that jointly learns word, sentence and document embeddings (see figure 3.4) called Doc2Sent2Vec [44]. It thereby applies specifically to documents rather than arbitrary textual units like PV.

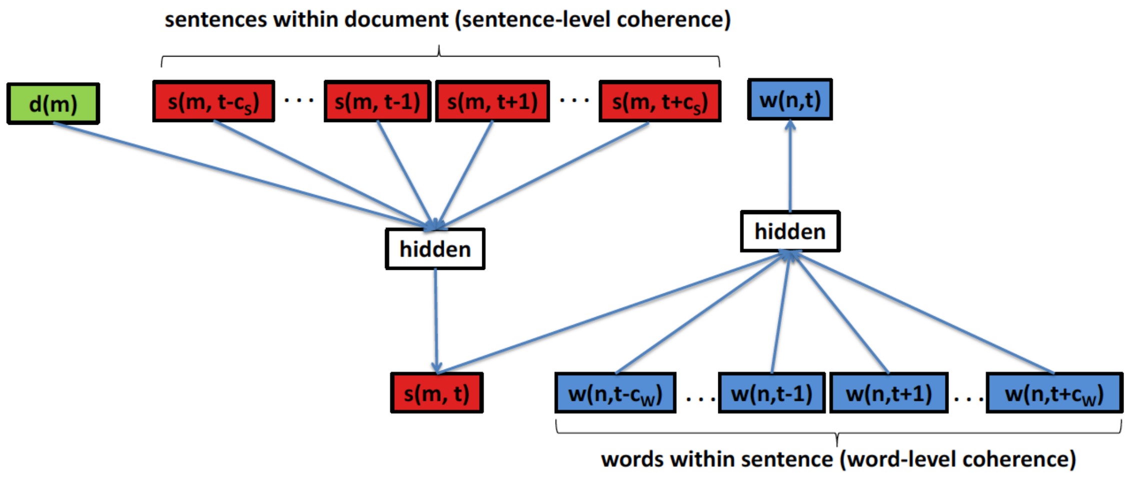


Figure 3.4: The architecture of Doc2Sent2Vec, using three types of embeddings (blue for words, red for sentences and green for the document) [44].

Doc2Sent2Vec can generally be described as a two-phase approach: In the first phase, sentence embeddings are learned using a word-level language model with contexts and a sentence token as input like in the original PV. The goal of that phase it to capture the topic for every single sentence in a low-dimensional form. In the second phase, the model learns document embeddings using a sentence-level language model based on the embeddings learned in the first phase. The total log likelihood probability for the Doc2Sent2Vec method thus defined as the sum of the log likelihood of both the words as well as the sentences occurring in a document:

$$L = L_{word} + L_{sent}$$

The fact that Dai et al. were able to show that jointly trained document embeddings tend to perform better on longer texts [31] and J et al. even improves on their results [44], makes this approach particularly promising for our case where documents can be expected to be of a rather large size.

3.3.1.3 Topic-LSTM

For the Topic-LSTM we use a basic LSTM architecture with the training objective of predicting the document's topic from a list of predefined topics. After reading all words of the input, the state of the LSTM will be used to infer the most probable topic of the observed document. Therefore, we consider this state as our document representation to be evaluated on the metrics described later in this chapter.

Similar to the Doc2(Sent2)Vec models, the Topic-LSTM takes word IDs from the vocabulary as input and projects them into a vector space using an embedding layer. This layer can be either initialized randomly or by using pretrained word embeddings. Also, the word embeddings can be further adjusted during training or kept static. One considerable difference though is that this type of NN can only handle inputs of fixed length. In our case, that means that every document the NN gets trained on has to have the same amount of words. As the average document length within our later described datasets is 419, we chose 400 as our desired input size. If a document happens to contain more than 400 words, all words beyond the 400th will be dropped. Should a document, on the other hand, contain less than 400, the missing number of words will be padded with $<PAD>$ tokens.

3.3.1.4 Topic-CNN

The Topic-CNN uses the same topic prediction training objective as the Topic-LSTM. Similarly, it also has an embedding layer as its first and classification layer as its last layer. It furthermore uses same fixed-length inputs of 400 words.

The main design choice of this NN architecture, however, lies in the convolutional layers. We decided to follow the same approach that Kim [49] uses in his work to perform several textual classification tasks on different kinds of datasets (none of which contains scientific publications). This network architecture uses three filters of different sizes which consider 3, 4 and 5 words. While increasing the filter sizes would intuitively provide a larger context and thus potentially better results, the literature shows that increasing the filter size actually does not yield any noticeable improvements [49]. These three filters will all slide over the matrix of word embeddings. The output of those filters are then concatenated (as shown in figure 3.5) and for the input to our classification layer and just like for the Topic-CNN also serve as document vectors to be evaluated on our metrics.

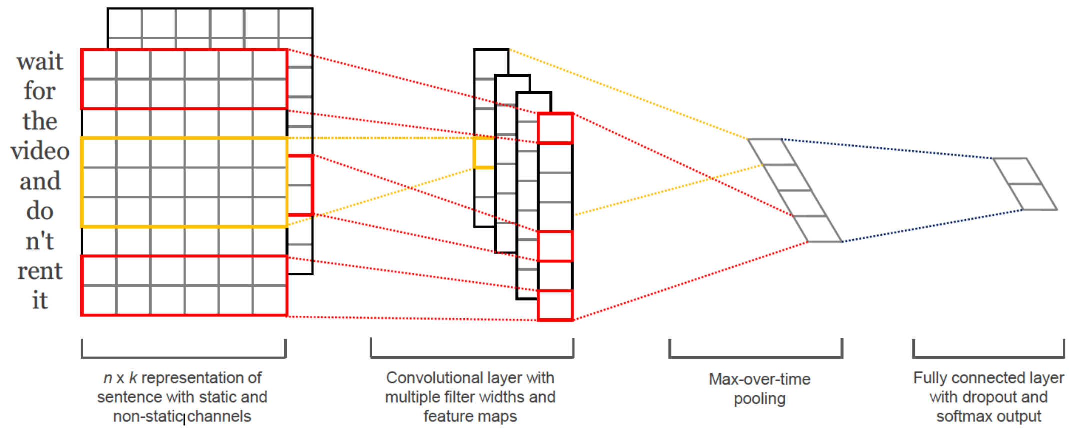

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 3.5: The CNN architecture used by Kim for several textual classification tasks [49].

3.3.1.5 Naive Autoencoder

As previously described in the theory section, the training objective of the autoencoder model is to reproduce the input document by encoding it into some lower dimensional embedding and therefrom decoding it back into the original format. While autoencoder architectures can be arbitrary complex, we decided to start with a rather simplistic variant: We use two fully connected layers for the encoding stage and another two fully connected layers for the decoding stage respectively. Additionally, we apply dropout after both the first encoding and decoding layers for regularization purposes.

As input to the NN, we use the mean of all word vectors of a document. These word vectors will be retrieved by using pretrained word embeddings identically to the models described above. This form of document representation will then be encoded by the first two fully connected layers resulting in a latent representation that we consider our document embedding. From this embedding, the decoding layers will then try to reconstruct the same vector used as input to the network. The loss function to optimize is, therefore, the mean squared error (MSE) between the input vector and the output vector.

3.3.1.6 Recurrent Autoencoder

As our last model, we will use a recurrent autoencoder which can be seen as a combination of the previously introduced RNN and autoencoder architectures. The goal of such NN is to reconstruct the input representation like the naive autoencoder but rather than processing the whole input document at once, it processes it word by word, changing its state and thereby taking into account information based on previously read words.

The main motivation to include this variant is to combine the LSTM's ability to process input sequentially but in an unsupervised fashion like the naive autoencoder, thus avoiding possible biases introduced by using the topic labels.

3.3.2 Setting

To be able to make a fair assessment regarding the performance of the evaluated approaches, the technical settings for all experimental runs must be fixed. Thus, all code will be run using the same machine configuration, to be precise a *c4.4xlarge* cloud instance provided by Amazon's AWS EC2 service[3].

Since different programming languages and the respective frameworks built with them usually show strong variances in performance, this factor shall also be fixed. All algorithms should thus be implemented in version 3.5 of the Python programming language, using Google's Tensorflow framework [7] for optimized numerical computations. One exception can hereby be made for the Keras framework [28]

[3]`https://aws.amazon.com/ec2/instance-types/`

which simply provides an abstraction layer over Tensorflow, using the same computational graph under the hood.

4

Results

While PV-DBOW has been evaluated on large corpora [31], Doc2Sent2Vec has only been assessed on rather small text corpora [44] not representative for document collections on the scale Iris AI works with. The RNN/LSTM and CNN approaches furthermore have not been evaluated on such kind of data at all which makes those two particularly interesting for our evaluation.

4.1 Hyperparameter Optimization

This section explains the results obtained for the conducted experiment described in 3.2. The first aspect we thereby consider is the number of epochs the model was trained on before stopping early due to a stagnation in its learning progress which can commonly be seen as an indicator of overfitting.

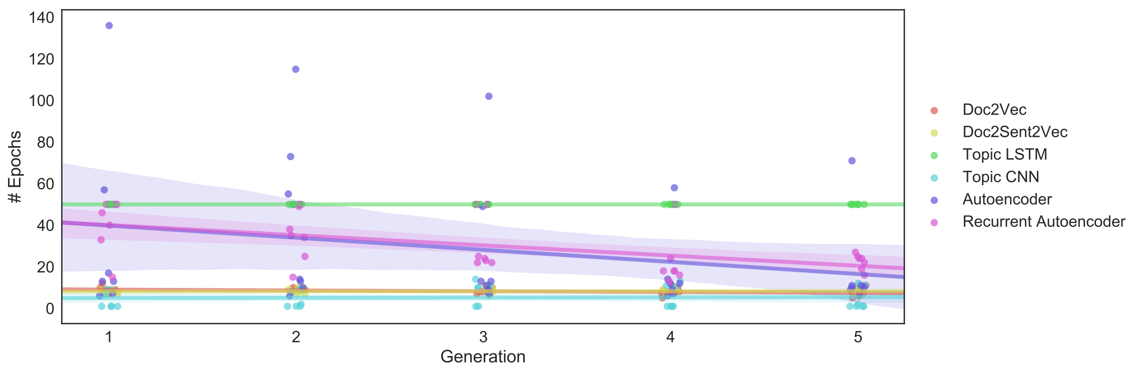


Figure 4.1: The number of epochs used by the models to train before reaching their maximum number or early stopping over all five generations.

Topic LSTM mostly uses its maximum of 50 epochs as can be seen in figure 4.1. This shows that the model has the potential to learn more. While we had to limit the number of epochs to 50 due to the long training time of LSTMs. In future experiments, this number should be increased by a considerable amount.

The naive autoencoder, as well as the recurrent autoencoder, show a decrease in the number of epochs before early stopping. This shows that the particle swarm optimization (PSO) indeed manages to find better hyperparameters combinations with each generation. Furthermore, a higher number of generations might have actually resulted in even better models.

Doc2Vec, Doc2Sent2Vec and the Topic CNN, on the other hand, all start overfitting within a few epochs. This, unfortunately, tells that the PSO is not able to improve the combination of hyperparameters beyond random guessing.

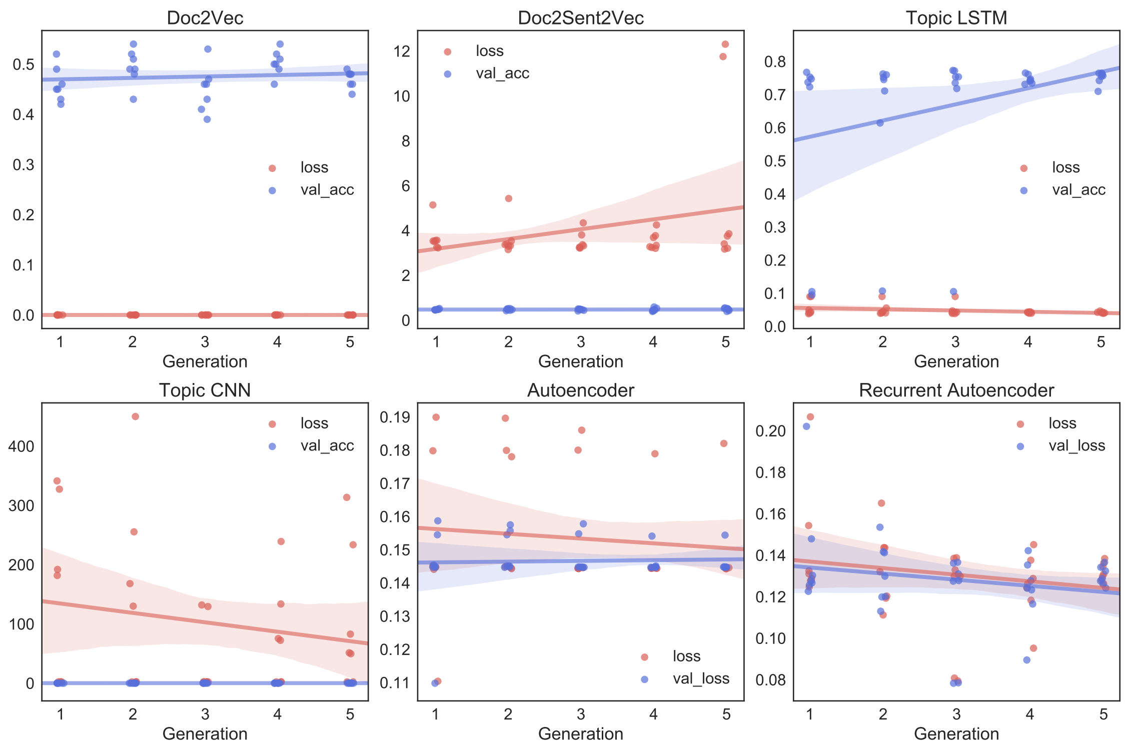


Figure 4.2: The test (red) and validations metrics (blue) used during model training grouped by generation.

Figure 4.2 illustrates how both Doc2Vec and Doc2Sent2Vec show no significant improvement in either loss or accuracy with an increasing number of generations. The trend line for the loss even increases in case of the Doc2Sent2Vec model. This furthermore strengthens our suspicion that no sufficient combination of hyperparameters could be found by running the PSO.

Similarly, the naive autoencoder does not show much variation either. While the training loss slightly decreases, the validation loss stays constant. This might be an indication that this model quickly reaches its limits without the ability to process text input sequentially like in case of RNNs (including LSTMs).

For the recurrent autoencoder, on the other hand, a general downward trend is visible. Analogous to our observation for the number of epochs, this shows us that the PSO is indeed able to find better-suited hyperparameters with increasing generations and thus results in better recurrent autoencoder models.

A similar development can be observed for the topic LSTM except that it is validated on accuracy which shows an upward trend, as a high accuracy is desirable. Nonetheless, the accuracy barely exceeds 80%.

The Topic CNN unfortunately, while still somewhat decreasing in loss, does not improve with regard to its validation accuracy.

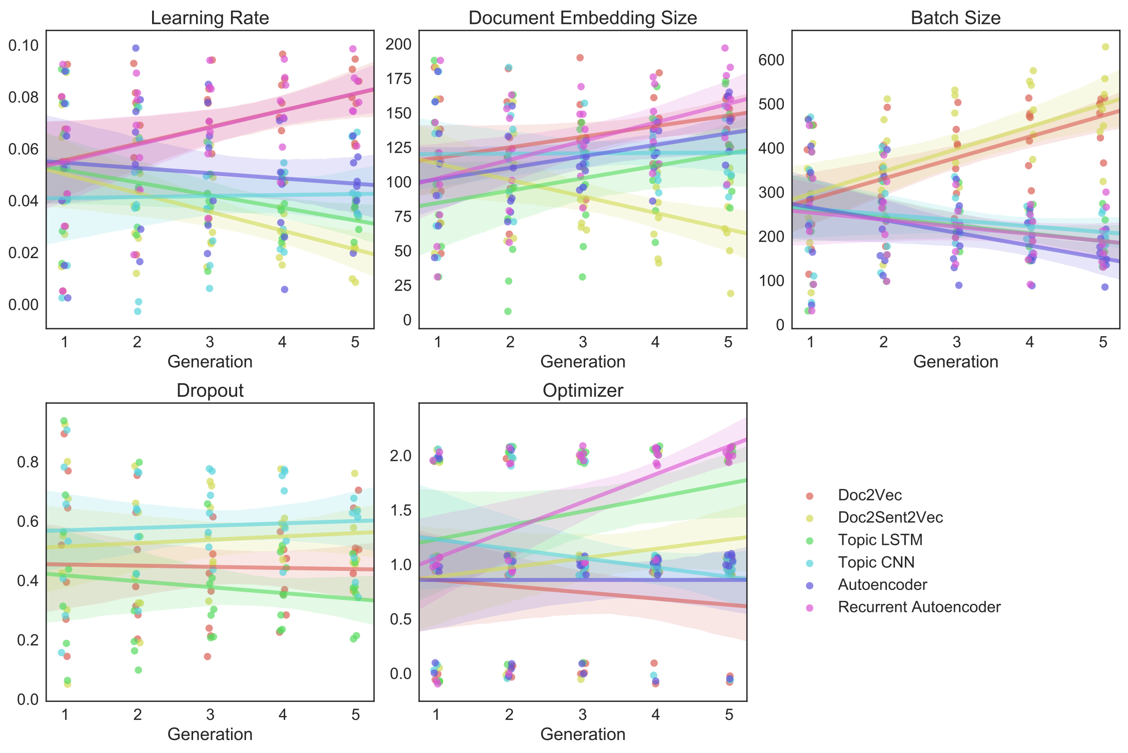


Figure 4.3: Change of hyperparameters during PSO for each model of the five generations. For the optimizer graph the y axis has to be read as follows: 0 = SGD, 1 = Adam, and 2 = AdaGrad.

Figure 4.3 illustrates how the different hyperparameters change for each model over the five generations. While only the Doc2Vec model moves towards a higher learning rate, the majority of models tends to work better with smaller learning rates. For the document embedding size, a general upward trend is observable with only the Doc2Sent2Vec approach seemingly performing better with lower dimensional document representations. When examining the batch size, two general groups can be observed: First, the correlation-based approaches, namely Doc2Vec and Doc2Sent2Vec tend to work better with larger batch sizes. Second, the rest of the approaches appear to perform better when trained in smaller batches. The dropout values stay more or less constant, which indicates, given the wide variance of values being used, that drop out does not have an essential influence on the models' performances in our case. Regarding the optimizer, it appears that for both of the LSTM models the Adagrad optimizer works best, while for the other model the ADAM optimizer leads to the best results.
When looking at figure 4.4, we notice that the scores for Doc2Vec and Doc2Sent2Vec lie barely above the random baseline. This is again not surprising when remember-

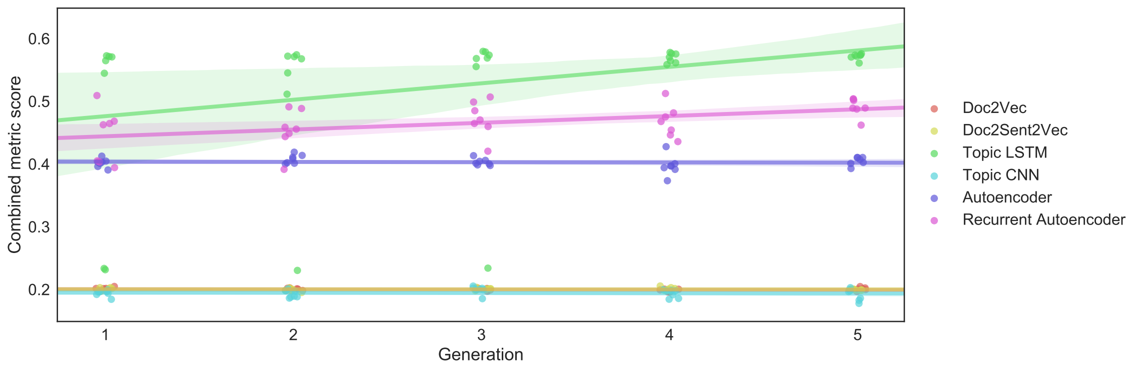


Figure 4.4: Change of the weighted sum of metrics for all approaches grouped by generation.

ing the generally bad performance of those approaches from earlier.

Equally, the Topic CNN has not been able to comprehend the documents' content either. While one could argue that the small filter size is not able to capture context sufficiently well, it seems more like the opposite is actually the case. Similar to Doc(2Sent)2Vec the CNN is not able to pick up all the nuances contained in the large vocabulary given the rather small training set.

While the Topic LSTM comes with quite high scores, we have to keep in mind that its performance is biased by good results on the topic classification metric. It is nonetheless interesting to see a general improvement during the PSO process.

The autoencoders perform surprisingly well considering they are completely unsupervised. While some lower scores can occasionally be observed, they occur only due to the insufficiently small learning rate in those cases. This indicates that we did not give the model enough epochs as it would have needed more time to fit to the training data.

4.2 CORE Dataset

4.2.1 Topic Prediction

Very unsurprisingly, the Topic LSTM performs best on the topic prediction metric as it was specifically trained on that task. The naive autoencoder on the other hand still scores reasonably high which was not to be expected considering it was not implicitly trained on the notion of a topic. The recurrent autoencoder comes in third, performing worse than both topic LSTM and the naive autoencoder. It is thereby particularly interesting to see that it performs best with the smallest word vector dimensionality. This gives rise to the suspicion that it would possibly need more epochs to learn from higher dimensional word embeddings.

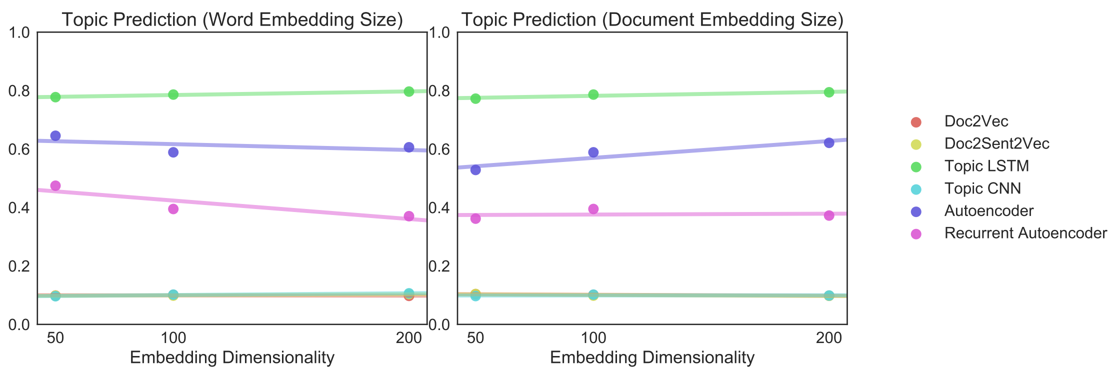


Figure 4.5: The accuracy on the topic prediction metric measured for different word embedding (left) and document embedding sizes (right).

4.2.2 Triplets

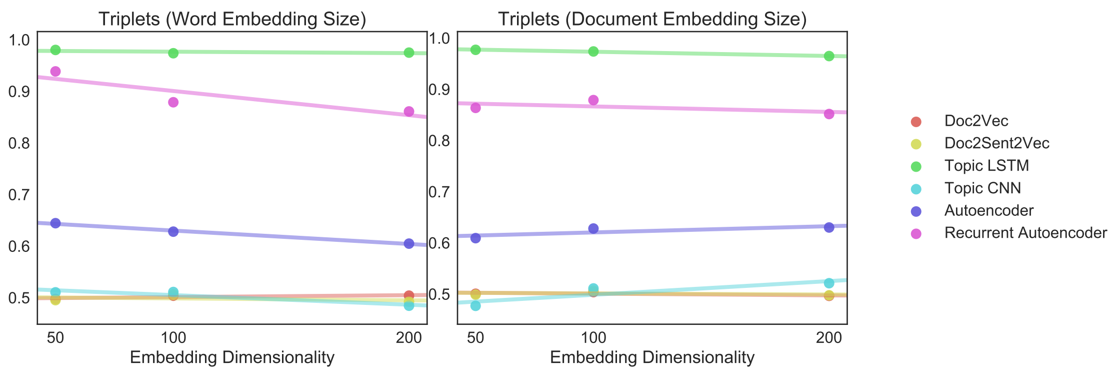


Figure 4.6: The accuracy on the triplets metric measured for different word embedding (left) and document embedding sizes (right).

Again, for the triplet metric, the Topic LSTM performs best. Unlike in case of the topic prediction, this can be once be regarded an actual success as the supervised training with topics cannot be seen as a bias on this metric. Rather it is interesting to note that training on topics seems to strongly improve the performance even with no topics used for inference.

Similarly strong is the recurrent autoencoder which comes in second with its best accuracy over 90%. Again this value has been achieved with the smallest word embedding dimensionality available.

4.2.3 Ranks

Both accuracy and precision happened to be unfortunately rather low for all approaches in this metric (compare figure 4.7). We did not include a graph for the precision here as its results do not differ significantly from the accuracy, but they can still be found in table A.1. Nonetheless, the naive autoencoder performs best

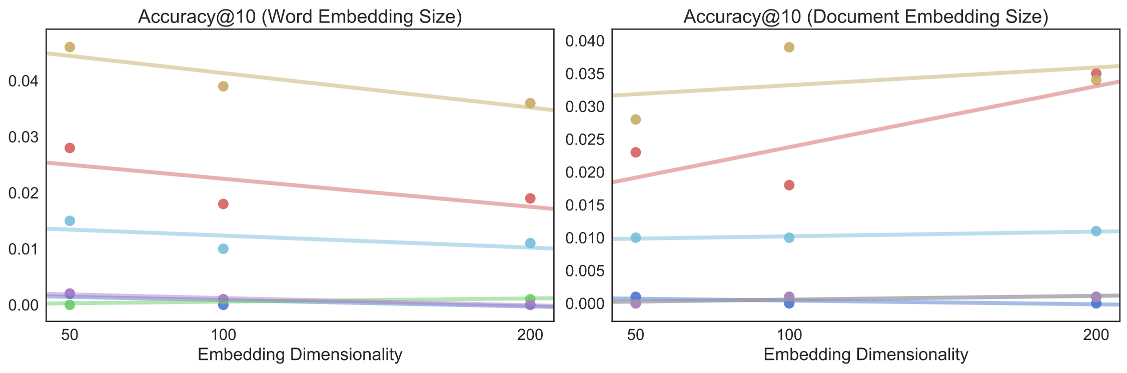


Figure 4.7: The accuracy@10 on the ranks metric measured for different word embedding (left) and document embedding sizes (right).

here. Also, we can generally observe that smaller embeddings both for words and for documents lead to a better performance.

4.2.4 Cluster Analysis

After evaluating the numerical results from the experiment, this section will take a deeper look at some characteristics of the created vector spaces. We, therefore, used the popular t-SNE algorithm [81] to transform the high dimensional document embeddings into two-dimensional space which is easier for humans to comprehend.

Out of our six approaches, we selected three examples that show interesting patterns of how documents can be clustered and visualized them in Figure 4.8. First, the dimensionally reduced vector space of documents for the Doc2Vec method shows no visible clustering at all which means that the t-SNE algorithm was not able to find any significant correlations in the original vector space. Second and in total opposite, the two-dimensional vector space for the Topic LSTM shows clearly separated clusters of topics with only a relatively small number of outliers. At last, the Autoencoder appears as one big cluster in which documents of the same topic still group together but without larger distances as observed for the Topic LSTM.

One thing to keep in mind is that the topics are not labeled by humans but rather retrieved from an algorithmically topic model. Therefore some documents that appear as outliers might actually be falsely assigned by the topic model in the first place. Also, many documents might actually belong to more than one topic. From this perspective, the more floating boundaries which can be observed for the (recurrent) autoencoder make more sense than the clear separations seen for the Topic LSTM which is trained on the assumption that each document belongs to exactly one topic.

Unfortunately, the clusters for the Doc2Vec, Doc2Sent2Vec and the Topic CNN, all shown in figure A.2, once again give an indication that those models are not able to capture the content of the documents.

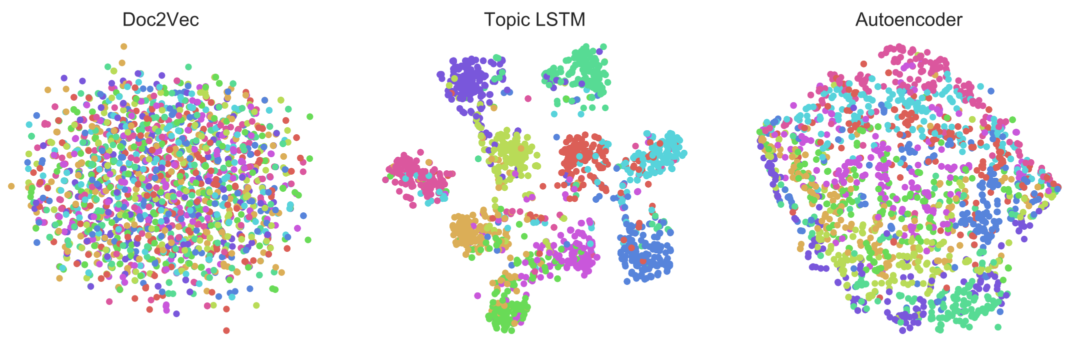
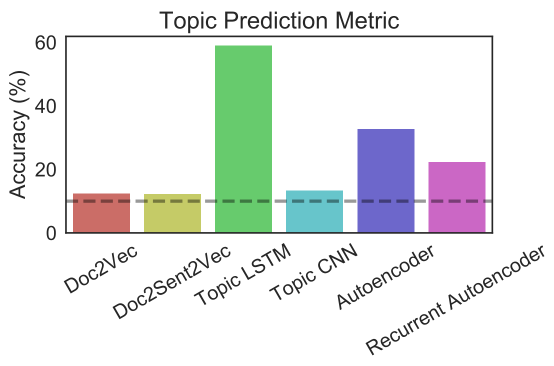
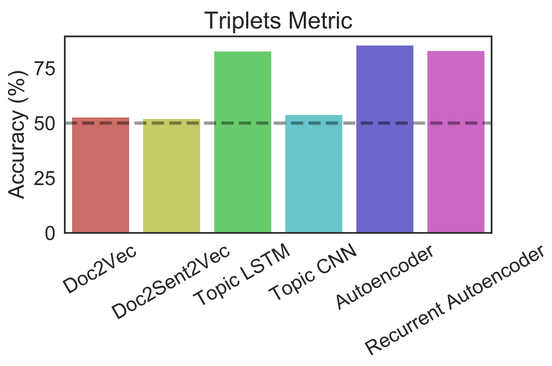


Figure 4.8: The document vector spaces for three selected models, reduced from several hundred to two dimensions using the t-SNE algorithm.



(a) The accuracy measured on the topic prediction metric.



(b) The accuracy measured on the triplets metric.

4.3 Autonomous Vehicle Dataset

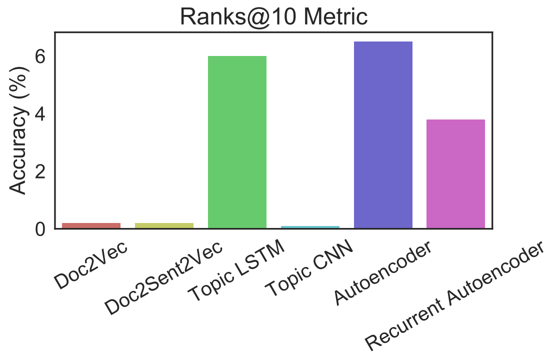
In the following, we present our experimental results for the AV5k10 datasets, mainly in order to analyze how well the performance of our approaches translates to other datasets.

4.3.1 Topic Prediction

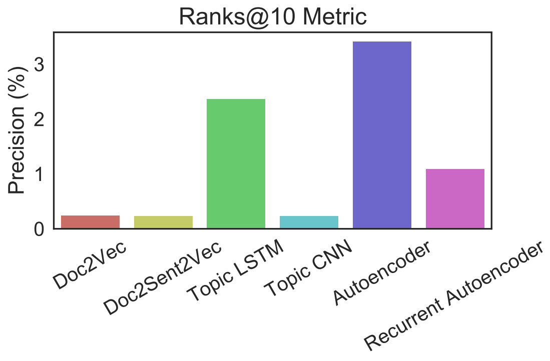
When looking at the results for the topic prediction metric, shown in figure 4.9a, we can clearly see that the accuracy of the Topic LSTM decreased by 20% and for the Naive Autoencoder and the Recurrent Autoencoder it even decreased by over 30%. After all the approaches do not change their order in performance.

4.3.2 Triplets

Similar to the topic prediction metric, all approaches that somewhat performed over the baseline do perform considerably worse compared to Core10k10 (see figure 4.9b). What is interesting to notice though, is that the naive autoencoder now performs



(a) The accuracy measured on the ranks@10 metric.



(b) The precision measured on the ranks@10 metric.

best of all approaches. This gives reason to suspect that neither topical nor sequential information is of significant importance when differentiating one document from another or at least to a smaller degree than for the Core10k10 dataset.

4.3.3 Ranks

While the performance for the topic prediction and triplets metrics considerably dropped, the ranks metric in contrast to that shows a slight increase in both accuracy and precision when compared to the Core10k10 dataset (see figures 4.9a and 4.9b). Similar to before, the naive autoencoder still scores best. In comparison to that, the Topic LSTM performs slightly better while the recurrent autoencoder performs worse. The reason for the increase in performance might thereby simply lie in the fact that the AV5k10 dataset is only half the size of Core10k10. Thus, the chance of choosing the right documents is naturally higher with fewer documents to choose from.

4.3.4 Time

While the computational complexity (of the model) of the Doc2Vec and Doc2Sent2Vec approaches grow with more (training) documents due to the size of the document embedding look-up matrix, all other models described in this work stay constant.

Additionally, the measured times shown in table A.1 indicate that for the approaches based on feed-forward NNs neither the size in word embeddings nor in document embeddings makes a difference. For the Topic CNN we see that while training and inference times increase with higher word embeddings, they actually stay constant for the document embedding size. This is due to the fact that for larger document embeddings the convolutional filters have to be applied a multitude of times. The document embeddings, on the other hand, are only used in the fully connected layer where having a matrix with 50, 100 or 200 rows makes no measurable difference. In contrast to that, we can observe that both LSTM approaches do actually take much longer with increasing embeddings sizes. The reason for this lies in the method's recurrent nature where the model's state, which has the same size as the resulting document embeddings, gets used over and over again, thus resulting in a linear time increase.

4.4 Mapping Study Application

After the previous sections have taken an in-depth look at the produced vector space models, this section will evaluate how well those models actually relate to the original problem of aiding in mapping studies. Currently, the automotive study uses a classical topic model for helping in differentiating the topics as shown in figure 4.9. Analogous to the ten most frequent topics we have used earlier, we also selected a topic model limited to ten topics. While some vector space models discussed in the previous sections build only very weak clusters (i.e. having a low perplexity), we chose the Topic LSTM for the following discussion as it showed the clearest clusters to be comparable with the clusters from the topic model.

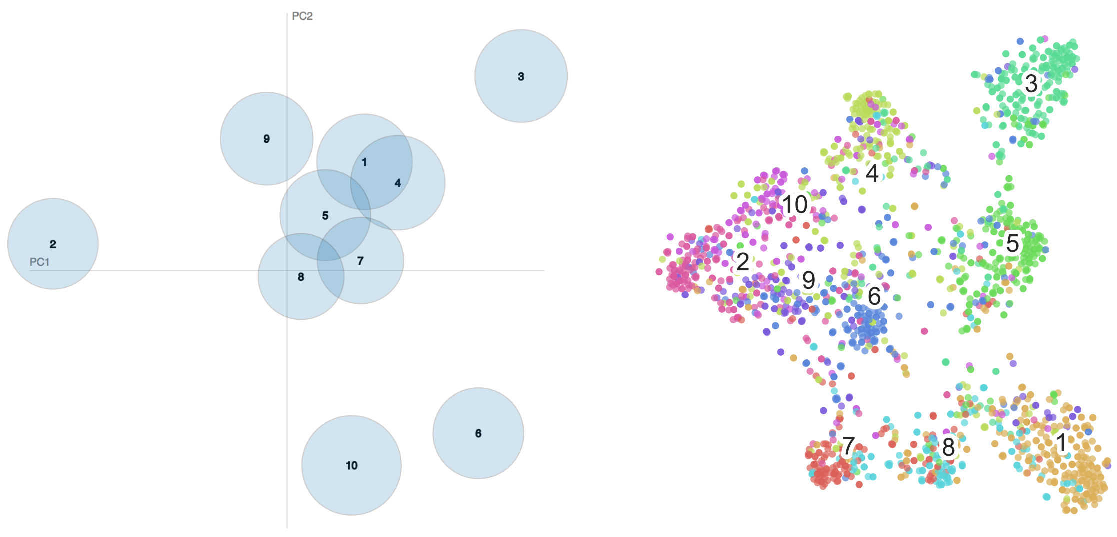


Figure 4.9: The topic clusters from the NTM (left) and our Topic LSTM model, reduced in dimensionality using the t-SNE algorithm. While the clusters of the NTM are represented as circles based on their averaged distribution in the two-dimensional space, for the Topic LSTM, we show all documents, colored by the topic which they belong to.

For the sake of better visualizing both the topic model as well as the vector space model, we reduced both into a two-dimensional space using the t-SNE algorithm [81] as already described in the previous sections. When comparing those t-SNE visualization, we can observe both similarities as well as several discrepancies, which are explained in the following.

To start with, topics *2* and *3* share the same position in both two-dimensional spaces. Also, they are really lying at the outer end of the two-dimensional space, showing a significant variance compared to the other topics.

On the other hand, for the mapping study, a lot of clusters are centered in the middle which indicates little variance between the topics. In case of the document vector space, topics are somewhat more spread which is not surprising since the underlying neural network optimizes inter-cluster distances during its training process. For the sake of the mapping study, clearly separated clusters can mostly be seen as preferable since they give a less noisy overview of the topics than several overlapping clusters like in case of the topic model.

Topic	**Top 5 Terms**	**Interpretation**
1	travel, travel time, times, travel times, regression	Travel time/route estimation
2	stability, adaptive, cruise, cruise control, platooning	Assisting with stability (e.g. cruise control)
3	single, measurement, compensation, satellite, receiver	Measurement sensors
4	spatial, temporal, flows, tree, estimation	Spacial/temporal mathematics
5	interacting, random, normal, distribution, plants	Probability theory, statistics
6	intersection, risk, exchange, infrastructure, management, communication	Intersection/pedestrian management
7	novel methods, detector, offline, investigation, macroscopic	Vehicle detection
8	road networks, microscopic, changes, road network, sensitivity	Navigating on roads
9	fuzzy, logic, fuzzy logic, controller, hierarchical	Logical systems
10	decision, making, policy, decision making, intelligent	Decision making

Table 4.1: Top 5 terms for the 10 most frequent topics of the AV dataset with manually added interpretations of the computationally retrieved topics.

One particularly interesting relation between two topics can be seen for 7 and 8: Topic 8 focuses more on solely navigating on the road, thinking more in lanes, also referred to as "microscopic". Topic 7, on the other hand, takes a more "macroscopic" view in the world, focusing not only on static environments but rather on other detecting other vehicles and their respective trajectory prediction. In case of the topic model, they lie in the center of the two-dimensional space, overlapping with other not particularly related topics. When considering the document vector space, on the other hand, both of them share the same value on the y-axis, indicating a semantic similarity. Furthermore, on the y-axis the are mirroring on 0, giving a hint that one might be the opposite of the other, exactly like "macroscopic" is the opposite of "microscopic".

While topic clusters are usually subjective to some degree, with no ground truth data available in most cases, we do still argue that our created document vector space provides several benefits compared to the conventional topic models, as we have demonstrated with the examples above. At last, even with an existing reliable topic model, it can still serve as a helpful method to get another view on how different topics relate to each other.

5

Discussion

5.1 Research Questions

RQ1.1 *What metric allows for a sound assessment of document embedding?*
By using a weighted sum of three different metrics we assure a fair comparison in which no method that only works well for one specific metric can succeed. The three used metrics themselves are widely explored and applied in both research and practice and can thus be assumed to provide a sound indicator for the quality of a document vector space.

RQ1.2 *What dataset(s) should be used that is not biased towards any method or task?*
To avoid bias towards specific domains, we used two inherently different datasets for our evaluation. While one is assembled with a rather small and specific topic in mind, the other one is much bigger and broader. This way we assure that our evaluation results apply to a variety of scenarios rather than just one specific dataset.

RQ2.1 *How well do different state of the art NNs for document embeddings perform on scientific publications?*
We found that the quality of the document vector space produced by the evaluated NNs varies to a great degree. While some produce good results, others perform barely better than the random baseline. Yet, the lack of good performance of the worse models might be an indicator that we, regardless of our hyperparameter optimization process, could not find optimal hyperparameters that could considerably boost the models' performance. Using PSO with a larger number of generations and over a larger number of hyperparameters will require a considerable amount of computational resources, but given those, will most likely result in a drastically improved performance.

RQ2.2 *How do parameters like the length of the text to be transformed or the document vector dimensionality affect the quality of the resulting embeddings?*
Our experiments show that it is not possible to make general claims on what values should be used for hyperparameters like document vector dimensionality. Instead, every method has its own set of optimal hyperparameters which have been rigorously analyzed in section 4 of this thesis.
Even though we only use abstracts as input, using the full textual document would also be an option which could unfortunately not be explored in this work due to the

limited time frame. The higher number of words of a full document compared to an abstract would result in increased model training times which make them subject to future follow-up research.

RQ2.3 *In what chunks (characters, words, etc.) should text be fed into the NN?*
In this work, we only explored word-based models in order to provide a fair evaluation where each model could use the same pre-trained word vectors. While character-based models have also shown promising results, we leave their evaluation up to future work.

5.2 Alternatives

Given the timeframe of this work, many alternative options could unfortunately not be explored with our evaluation framework.

One quite popular model is so-called Skip-Thought Vectors [51] for which Google recently open sourced an implementation [1], unfortunately not in time to be included in this work.

Another common but non-neural option is the utilization of so-called centric approaches [74]. Like the name suggests, they build document representations as the average (i.e. the geometric center) of a sentences or documents word embeddings. This leads to the problem that such models are heavily dependent on the quality of the utilized word embeddings. On the other side, they usually possess no or few hyperparameters which makes the tedious task of hyperparameter tuning obsolete.

5.3 Threats to Validity

The arguably biggest threat to validity in this work is the fact that the proposed metrics are all not based on human labeled ground truth but rather algorithmically. Because manually labeled data on this scale could not be retrieved, we had to generate topics using the neural topic model and distances for the triplet and ranks metrics based on the WISDM algorithm. On the other hand, we have to remember that even human labeled data in this domain is subjective and therefore no ideal solution. We, therefore, argue that while not being ideal, our ground truth is still reasonably accurate to produce valid results when used in accordance with our evaluation framework. The fact that some models perform far better than the baseline while barely match the baseline gives further indication of the validity of our claim.

Another point of critique our evaluation might have to take is its insufficient exploration of the hyperparameter space. This could as earlier described be solved by having more time and computational resources. Given these resources, the performance of Doc2Vec, Doc2Sent2Vec and the Topic CNN shown in other papers

[1] `https://github.com/tensorflow/models/tree/master/skip_thoughts`

might be reproducible on the CORE and autonomous vehicles datasets once the right hyperparameters can be found.

5.4 Future Work

The first apparent step for future work should be running experiments using the larger datasets. Since we already curated the Core250k10 and Core1M+100 datasets, those would naturally be the ones to use. While the smaller datasets could be seen as a first indication what approaches and respective hyperparameters for them to work, the larger datasets will most likely give more practical insights on how the vector space models perform on real-world-scale data, both in terms of metric performance as well as time and space requirements.

One reason why the models did not perform so well might be the quality of the word embeddings since all models represent their input as a list of words embeddings and thus heavily rely on their quality. New methods to build improved word representations by taking into account sub-word information, like done by Facebook's FastText [21, 45] or training the network on some translation task before the actual task to compute the embeddings like the in case of Salesforce Einstein's Context Vectors (CoVe) [62] are therefore promising candidates to apply to document vector space approaches. Alternatively, word embeddings could be skipped in the first place and substituted with character embeddings which pose an elegant way to deal with large vocabularies for which a word embedding for every word would have to exist.

Finally, there is currently a stream of new conferences and workshops on the topic emerging. For example, the RepL4NLP (Workshop on Representation Learning for NLP)[2] was initiated in 2016, while the RepEval [3], a workshop on evaluating vector space representations for NLP took place for the first time this year. This indicates that there is a large amount of novel research to come in the future.

[2]`https://sites.google.com/site/repl4nlp2017/home`
[3]`https://repeval2017.github.io/`

6 Conclusion

As the first part of this thesis, we propose a novel evaluation framework for vector space models in the domain of scientific publications. As part of this, we curated four datasets for two substantially different cases. Furthermore, we created three metrics for the models to be evaluated on. By publishing this framework, we invite other researchers and practitioners to also evaluate their approaches in the future.

As the second part, we used the earlier described framework to run experiments in order to evaluate six state of the art neural network based approaches to create vector space models. For this to be possible we also implemented those approaches in Python using Tensorflow and Keras. In addition, we included a detailed description of our models, from their architecture down to the choice of hyperparameters. This makes it easy to reproduce our results and evaluate own approaches in any programming language or framework. As a result, we could observe that LSTMs perform really well, followed by autoencoders regardless of the lack of labeled data. Especially the recurrent autoencoder should be further researched as it is especially appealing due to its unsupervised nature which is really valuable in a domain where no truly unbiased ground truth data exists. Also, both architecture and hyperparameters should be tuned further so that the approach matches the performance of the naive autoencoder for the cases where the latter scores better. Doc2(Sent2)Vec and CNN, on the other hand, were not able to capture document semantics in our case of scientific publications. Nonetheless, these approaches might still be able to perform considerably better in case better hyperparameters can be found or through some modifications to the architecture that we are currently not aware of. Both of the latter problems could most likely be solved by a more extensive hyperparameter optimization, which was unfortunately not possible with our available resources, but is strongly recommended for future work.
At last, we demonstrated how our vector space model can be utilized to aid in systematic mapping studies. Compared to classic topic models, it can help in discovering more meaningful semantic relations between topics. Also, when used together with such topics models, it will provide additional insights that might not have been visible from the topic model.

Bibliography

[1] ASE 2017. http://www.ase2017.org/. Accessed: 2017-02-23.

[2] Classic3 and Classic4 DataSets. http://www.dataminingresearch.com/index.php/2010/09/classic3-classic4-datasets. Accessed: 2017-02-19.

[3] CORE. https://core.ac.uk/. Accessed: 2017-02-23.

[4] Nearest Neighbor Search - Wikipedia. https://en.wikipedia.org/wiki/Nearest_neighbor_search. Accessed: 2017-02-14.

[5] SSBSE'17. http://ssbse17.github.io/. Accessed: 2017-02-23.

[6] Vision – Iris AI - Your Science Assistant. https://iris.ai/vision/. Accessed: 2017-01-16.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2015.

[8] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel M. Cer, Mona T. Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Iñigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *SemEval@NAACL-HLT*, 2015.

[9] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel M. Cer, Mona T. Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. Semeval-2014 task 10: Multilingual semantic textual similarity. In *SemEval@COLING*, 2014.

[10] Eneko Agirre, Carmen Banea, Daniel M. Cer, Mona T. Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *SemEval@NAACL-HLT*, 2016.

[11] Eneko Agirre, Daniel M. Cer, Mona T. Diab, and Aitor Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In *SemEval@NAACL-HLT*, 2012.

[12] Eneko Agirre, Daniel M. Cer, Mona T. Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *sem 2013 shared task: Semantic textual similarity. In **SEM@NAACL-HLT*, 2013.

[13] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51:117–122, 2006.

[14] Kushal Arora. Contrastive perplexity: A new evaluation metric for sentence level language models. *CoRR*, abs/1601.00248, 2016.

[15] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.

[16] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*, 2014.

[17] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW*, 2005.

[18] Terry Van Belle and David H. Ackley. Code factoring and the evolution of evolvability. In *GECCO*, 2002.

[19] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2000.

[20] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[21] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[22] David Budgen, Emilia Mendes, and Giuseppe Visaggio. Guidelines for performing systematic literature reviews in software engineering. 2007.

[23] Ziqiang Cao, Sujian Li, Yang Liu, Wenjie Li, and Heng Ji. A novel neural topic model and its supervised extension. In *AAAI*, 2015.

[24] Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. Improving multi-document summarization via text classification. *CoRR*, abs/1611.09238, 2016.

[25] Tanmoy Chakraborty, Sandipan Sikdar, Vihar Tammana, Niloy Ganguly, and Animesh Mukherjee. Computer science fields as ground-truth communities: their impact, rise and fall. In *ASONAM*, 2013.

[26] Jonathan Chang, Jordan L. Boyd-Graber, Sean Gerrish, Chong Wang, and David M. Blei. Reading tea leaves: How humans interpret topic models. In *NIPS*, 2009.

[27] Billy Chiu, Anna Korhonen, and Sampo Pyysalo. Intrinsic evaluation of word vectors fails to predict extrinsic performance. 2016.

[28] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[29] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evolutionary Computation*, 6:58–73, 2002.

[30] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011.

[31] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *CoRR*, abs/1507.07998, 2015.

[32] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41:391–407, 1990.

[33] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

[34] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. *CoRR*, abs/1605.02276, 2016.

[35] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20:116–131, 2001.

[36] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, 2016.

[37] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck. Contextual lstm (clstm) models for large scale nlp tasks. *CoRR*, abs/1602.06291, 2016.

[38] Anna Gladkova and Aleksandr Drozd. Intrinsic evaluations of word embeddings: What can we do better? 2016.

[39] Z. Harris. Distributional structure. *Word*, 1954.

[40] Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In *ACL*, 2013.

[41] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41:665–695, 2015.

[42] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[43] Gao Huang, Chuan Guo, Matt J. Kusner, Yu Sun, Fei Sha, and Kilian Q. Weinberger. Supervised word mover's distance. In *NIPS*, 2016.

[44] Ganesh J, Manish Gupta, and Vasudeva Varma. Doc2sent2vec: A novel two-phase approach for learning document representation. In *SIGIR*, 2016.

[45] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[46] Dan Jurafsky. Probabilis1c language models. In *Stanford*, 2017.

[47] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

[48] James Kennedy and Russell Eberhart. Particle swarm optimization - neural networks, 1995. proceedings., ieee international conference on. 1995.

[49] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[50] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[51] Jamie Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *NIPS*, 2015.

[52] Barbara A. Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen G. Linkman. Systematic literature reviews in software engineering - a systematic literature review. *Information & Software Technology*, 51:7–15, 2009.

[53] Barbara A. Kitchenham, Rialette Pretorius, David Budgen, Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen G. Linkman. Systematic literature reviews in software engineering - a tertiary study. *Information & Software Technology*, 52:792–805, 2010.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[55] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *ICML*, 2015.

[56] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, 2015.

[57] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[58] Yann Lecun, L Eon Bottou, Yoshua Bengio, and Patrick Haaner. Gradient-based learning applied to document recognition. 1998.

[59] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225, 2015.

[60] Tal Linzen. Issues in evaluating semantic spaces using word analogies. *CoRR*, abs/1606.07736, 2016.

[61] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. 2008.

[62] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *arXiv preprint arXiv:1708.00107*, 2017.

[63] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[64] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[65] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, 2013.

[66] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36:2227–2240, 2014.

[67] Neha Nayak and Christopher D. Manning. Evaluating word embeddings using a representative suite of practical tasks. 2016.

[68] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. 2004.

[69] Stephen M Omohundro. Five balltree construction algorithms. 1989.

[70] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[71] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *EASE*, 2008.

[72] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, 1998.

[73] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[74] Y. Liang S. Arora and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.

[75] Ruslan Salakhutdinov and Geoffrey E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50:969–978, 2009.

[76] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24:513–523, 1988.

[77] Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *EMNLP*, 2015.

[78] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

[79] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[80] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. *CoRR*, abs/1508.00200, 2015.

[81] Laurens van der Maaten, Geoffrey Hinton, and Yoshua Bengio. Visualizing data using t-sne. 2008.

[82] Suhang Wang, Jiliang Tang, Charu C. Aggarwal, and Huan Liu. Linked document embedding for classification. In *CIKM*, 2016.

[83] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015.

[84] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

A

Appendix

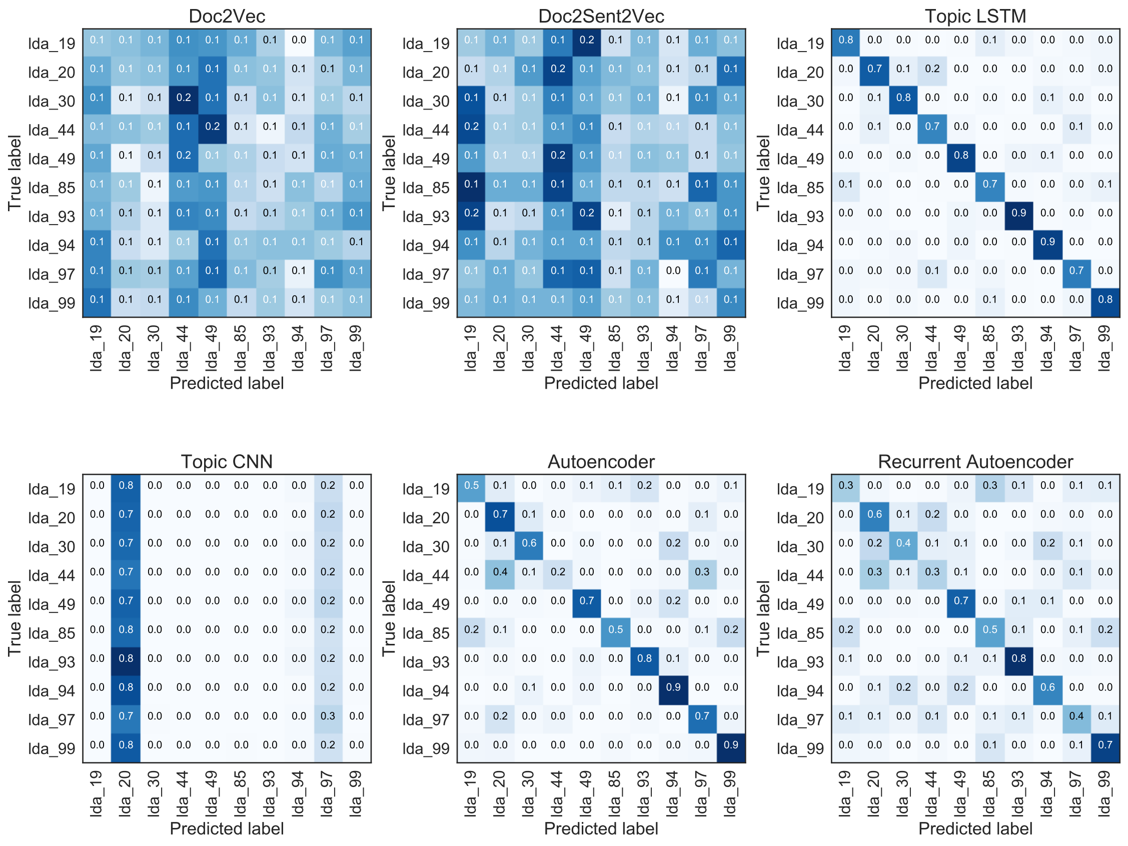


Figure A.1: The confusion matrices for predicting topics using the Core10k10 dataset.

Model	Topic Acc	Triplet Acc	Ranks@10 Acc	Ranks@10 Prec	Average	Inference Time	Training Time
autoencoder-100wv-100dv.json	58.88%	62.80%	3.90%	2.56%	41.64%	13 sec	56 sec
autoencoder-100wv-137dv.json	62.76%	63.42%	2.00%	2.43%	42.80%	13 sec	62 sec
autoencoder-100wv-200dv.json	62.12%	63.00%	3.40%	2.48%	42.69%	13 sec	68 sec
autoencoder-100wv-50dv.json	52.85%	60.92%	2.80%	2.51%	38.81%	13 sec	54 sec
autoencoder-200wv-137dv.json	60.61%	60.48%	3.60%	2.40%	41.36%	23 sec	95 sec
autoencoder-50wv-137dv.json	64.55%	64.44%	4.60%	2.89%	44.24%	8 sec	46 sec
doc2sent2vec-100wv-100dv.json	9.85%	50.65%	0.10%	0.09%	20.20%	0 sec	55.95 min
doc2sent2vec-100wv-200dv.json	9.79%	49.76%	0.10%	0.05%	19.87%	0 sec	68.43 min
doc2sent2vec-100wv-50dv.json	10.41%	49.89%	0.00%	0.07%	20.11%	0 sec	60.69 min
doc2sent2vec-100wv-96dv.json	9.05%	49.93%	0.20%	0.13%	19.71%	0 sec	62.16 min
doc2sent2vec-200wv-96dv.json	10.35%	49.23%	0.10%	0.06%	19.89%	0 sec	98.48 min
doc2sent2vec-50wv-96dv.json	9.85%	49.52%	0.00%	0.11%	19.81%	0 sec	49.20 min
doc2vec-100wv-100dv.json	10.09%	50.38%	0.00%	0.07%	20.17%	0 sec	8.55 min
doc2vec-100wv-138dv.json	9.56%	49.46%	0.10%	0.09%	19.71%	0 sec	13.46 min
doc2vec-100wv-200dv.json	9.82%	49.65%	0.00%	0.10%	19.84%	0 sec	10.18 min
doc2vec-100wv-50dv.json	10.35%	50.06%	0.10%	0.16%	20.18%	0 sec	8.94 min
doc2vec-200wv-138dv.json	9.76%	50.40%	0.00%	0.10%	20.07%	0 sec	10.74 min
doc2vec-50wv-138dv.json	9.86%	49.75%	0.20%	0.06%	19.91%	0 sec	11.90 min
rae-100wv-100dv.json	39.48%	87.88%	1.00%	0.68%	42.73%	112 sec	26.30 min
rae-100wv-176dv.json	39.64%	87.08%	0.90%	0.63%	42.49%	4.01 min	38.13 min
rae-100wv-200dv.json	37.24%	85.16%	1.10%	0.58%	41.08%	4.56 min	42.60 min
rae-100wv-50dv.json	36.18%	86.36%	1.00%	0.59%	41.11%	75 sec	19.06 min
rae-200wv-176dv.json	37.03%	86.10%	1.10%	0.67%	41.34%	4.65 min	58.54 min
rae-50wv-176dv.json	47.45%	93.86%	1.50%	1.07%	47.53%	3.69 min	28.83 min
topic_cnn-100wv-100dv.json	10.18%	51.10%	0.10%	0.04%	20.45%	84 sec	69 sec
topic_cnn-100wv-126dv.json	9.94%	49.00%	0.10%	0.12%	19.68%	83 sec	70 sec
topic_cnn-100wv-200dv.json	9.88%	52.12%	0.10%	0.12%	20.70%	84 sec	70 sec
topic_cnn-100wv-50dv.json	9.70%	47.70%	0.00%	0.05%	19.14%	84 sec	59 sec
topic_cnn-200wv-126dv.json	10.64%	48.44%	0.00%	0.07%	19.70%	160 sec	91 sec
topic_cnn-50wv-126dv.json	9.64%	51.04%	0.20%	0.15%	20.28%	45 sec	45 sec
topic_lstm-100wv-100dv.json	78.64%	97.38%	1.80%	1.92%	59.29%	100 sec	325.83 min
topic_lstm-100wv-143dv.json	79.24%	97.24%	1.60%	2.02%	59.43%	139 sec	470.78 min
topic_lstm-100wv-200dv.json	79.42%	96.48%	3.50%	2.09%	59.57%	4.41 min	679.42 min
topic_lstm-100wv-50dv.json	77.24%	97.70%	2.30%	1.96%	59.02%	63 sec	163.78 min
topic_lstm-200wv-143dv.json	79.67%	97.50%	1.90%	1.82%	59.68%	163 sec	560.75 min
topic_lstm-50wv-143dv.json	77.76%	98.02%	2.80%	2.01%	59.39%	128 sec	428.82 min

Table A.1: Experiment results from using the Core10k10 dataset.

	Topic Acc	Triplet Acc	Ranks@10 Acc	Ranks@10 Prec	Average	Inference Time	Training Time
av-autoencoder-50wc-137dv	32.67%	85.21%	6.50%	3.41%	40.95%	7 sec	36 sec
av-doc2sent2vec-100wc-100dv	12.23%	51.87%	0.20%	0.23%	21.44%	0 sec	89.19 min
av-doc2vec-100wc-50dv	12.32%	52.39%	0.20%	0.24%	21.64%	0 sec	4.84 min
av-rae-50wc-176dv	22.22%	82.66%	3.80%	1.09%	35.78%	114 sec	14.52 min
av-topic_cnn-100wc-200dv	13.29%	53.57%	0.10%	0.23%	22.34%	42 sec	59 sec
av-topic_lstm-100wc-200dv	58.94%	82.54%	6.00%	2.36%	48.55%	110 sec	372.55 min

Table A.2: Experiment results from using the AV5k10 dataset.

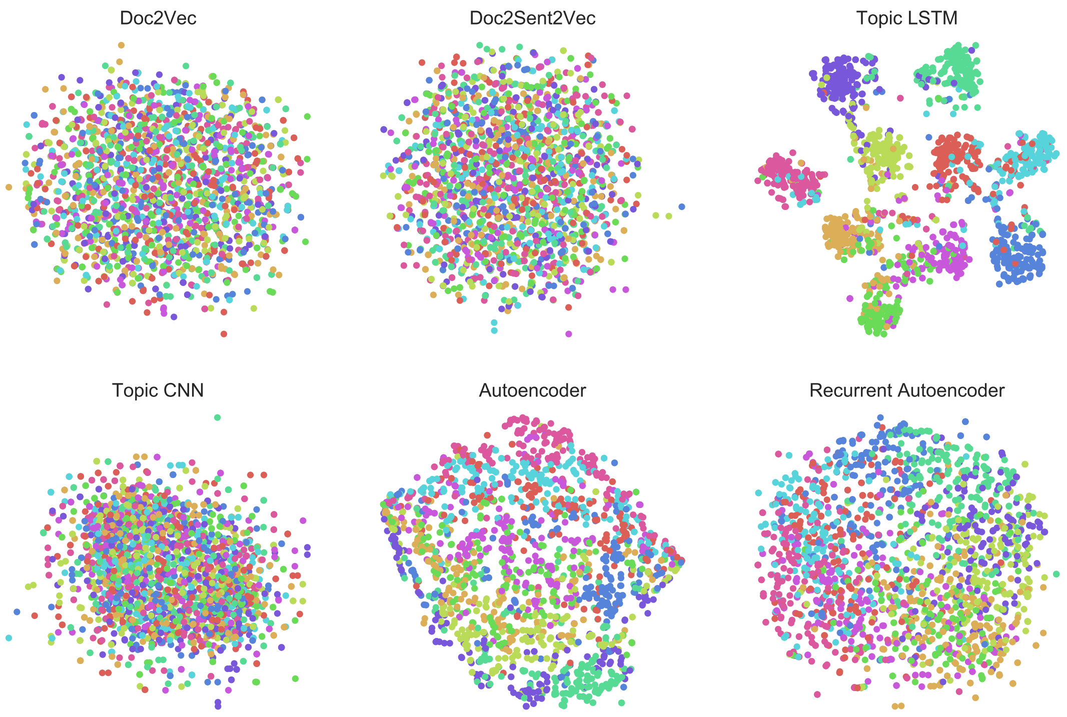


Figure A.2: The document vector spaces for the Core10k10 dataset reduced from several hundred to two dimensions using the t-SNE algorithm.

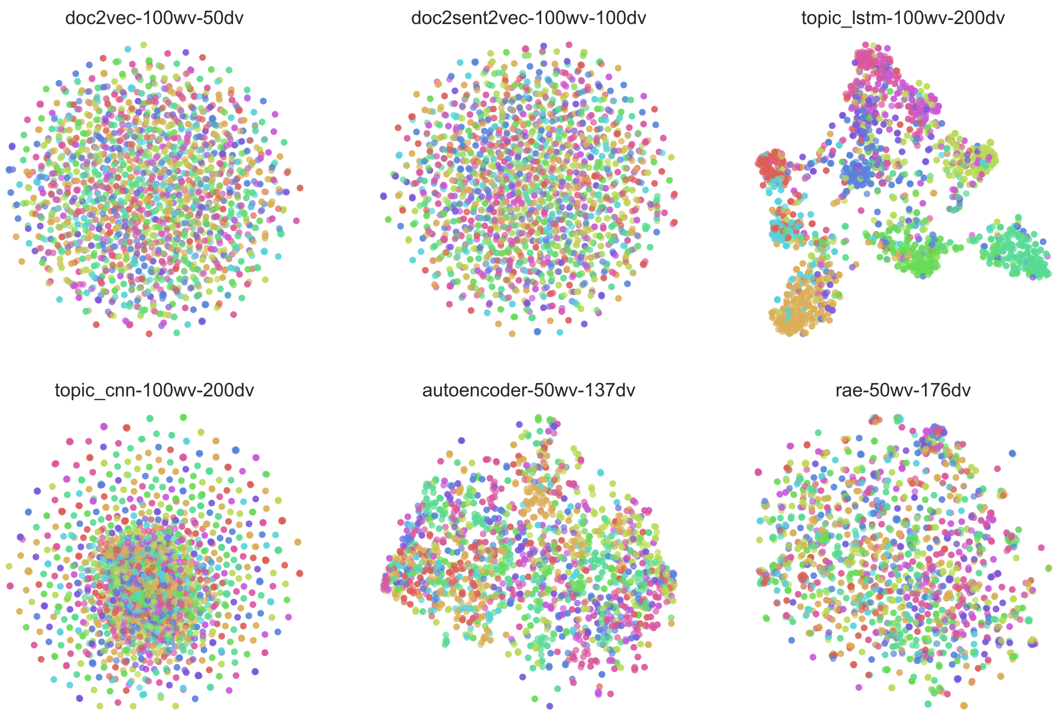


Figure A.3: The document vector spaces for the AV5k10 dataset reduced from several hundred to two dimensions using the t-SNE algorithm.